



WebApp Secure 5.1.3



Published: 2013-11-20

Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Copyright © 2013, Juniper Networks, Inc. All rights reserved.

Juniper Networks, Junos, Steel-Belted Radius, NetScreen, and ScreenOS are registered trademarks of Juniper Networks, Inc. in the United States and other countries. The Juniper Networks Logo, the Junos logo, and JunosE are trademarks of Juniper Networks, Inc. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

WebApp Secure 5.1.3

Copyright © 2013, Juniper Networks, Inc.
All rights reserved.

The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at <http://www.juniper.net/support/eula.html>. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

	About the Documentation	xix
	Documentation and Release Notes	xix
	Documentation Conventions	xix
	Documentation Feedback	xxi
	Requesting Technical Support	xxi
	Self-Help Online Tools and Resources	xxii
	Opening a Case with JTAC	xxii
Part 1	Overview	
Chapter 1	WebApp Secure	3
	WebApp Secure Summary	3
	How It Works	3
	Four Phases of Detection and Response	4
	WebApp Secure Features	4
	WebApp Secure Services	5
	Limitations	6
	WebApp Secure Appliance Terminology	6
	Using WebApp Secure with Third-Party Load Balancer	6
Chapter 2	Deployment	7
	Appliance Deployment Overview	7
	Placement Between Firewall and Web Servers	7
	Options for Load-Balanced Environments	9
	SSL Traffic Considerations	9
	EC2 Deployment	10
	Using a Secure Cluster	10
Chapter 3	Configuration	11
	First Time Configuration	11
	Web Interface Configuration Overview	11
	About the Configuration Wizard	12
	Basic vs. Expert Configuration	12
	Using the CLI	13
Chapter 4	Clustering	15
	Using a Secure Cluster	15
	Node Types	15
Chapter 5	High Availability	17
	High Availability Overview	17

Chapter 6	SRX Integration	19
	Integration with SRX Overview	19
Chapter 7	Appliance Management	21
	Appliance Management Overview	21
Chapter 8	Spotlight Secure	23
	About Spotlight Secure	23
Chapter 9	The Processors	25
	Processors Overview	25
Chapter 10	Reports	27
	Reporting Overview	27
	Scheduling a Report Overview	27
Part 2	Configuration	
Chapter 11	Initial Tasks	31
	Initial Appliance Configuration	31
	TUI Steps	31
	Setting the Hostname	32
	Interface Configuration	33
	Set DNS	34
	Restart the network	35
	Initialize the Appliance	35
	Verify Connectivity	36
	Install the License	37
Chapter 12	Configuration Wizard	39
	Configuration Wizard: Backend Server	39
	Backend Server Requirements	39
	SMTP Servers	40
	Wizard Alert Service	41
	Email Alert Requirements	42
	Wizard Backup Service	43
	Backup Service Fields	44
	Wizard: Spotlight Secure	44
	Wizard Confirmation Page	44
Chapter 13	Command Line Interface	47
	Using the CLI	47
	CLI: Config	48
	CLI: Config: Setting a Configuration Parameter	48
	CLI: Config: Initializing the Configuration	49
	CLI: Config: Import/Export	49
	CLI: Config: Configure a Proxy Exclusion	50
	CLI: Services	50
	CLI: System	50

Chapter 14	Configuration Options	51
	Securing Multiple Web Servers	51
	Create a New Application	51
	Edit Applications	52
	Application Patterns	53
	Define Backend Servers	55
	Enable SSL to the Client	56
	Whitelist Settings	57
	Configure Support for Akamai Dynamic Site Accelerator	58
Chapter 15	Clustering	61
	Using a Secure Cluster	61
	Setting Up Clustering	61
	Updating the Cluster	63
Chapter 16	High Availability	65
	High Availability Overview	65
	Configuring High Availability	65
	Updating with High Availability	69
Chapter 17	SRX Integration	71
	Filters and Terms Configuration Summary for SRX Integration	71
	Creating SRX Filters and Terms	72
	Configure the SRX Integration	73
Chapter 18	Reports	77
	Schedule a Report	77
	The Reports CLI	79
	Supported arguments	80
	Data Sources	81
	Formatting	81
	Example Report	82
Part 3	Administration	
Chapter 19	General Tasks	85
	Changing the Password	85
	Resetting the Password	85
Chapter 20	Verify	87
	Verify the Installation	87
Chapter 21	EC2 Deployment	89
	EC2: Deploying Using the Command Line	89
	EC2: Deploying Using the Web Interface	90
	Assigning the Instance and IP Using the CLI	94
	Assigning the Instance and IP Using the Web Interface	94
	Verify the Instance is Running	95
Chapter 22	Configuration Modes and Roles	97
	Basic Configuration Mode	97
	Expert Configuration Mode	98

	Role-Based Administrator Access Control	99
	Configuring Role-Based Access Control	99
	RBAC Groups and Roles	100
	Edit User Preferences	103
	Unblock Login Ban	104
Chapter 23	SRX Integration	107
	Testing the SRX Integration Configuration	107
Chapter 24	Appliance Management	109
	Restart and Shutdown the Appliance	109
	System Updates	109
	Statistics	112
	Master - Slave Mode	115
	Managing Services	115
	Backup and Recovery Overview	115
	Health Check URL	116
	Backup and Recovery Overview	117
	Restoring a Backup	117
Chapter 25	Spotlight Secure	119
	Enabling Spotlight Secure	119
Chapter 26	Security Monitor	123
	The Dashboard	123
	Attackers	128
	Responses Tab	131
	Locations Tab	132
	Incidents	133
	Counter Responses	135
	Sessions	136
	Search	137
	Reporting	139
	Configuration	139
	System Status	140
	Updates	140
Chapter 27	Autoresponse Defaults and Rule Creation	143
	Autoresponse Overview	143
	Editor Overview	147
Part 4	Monitoring	
Chapter 28	The Processors	151
	Complexity Rating Definitions	151
	Security Engine Incidents	152
	Session Cookie Spoofing	152
	Session Cookie Tampering	152
	Security Processors	153

Chapter 29	Honeybot Processors	155
	Honeybot Processors: Access Policy Processor	156
	Honeybot Processors: Access Policy Processor: Incidents - Malicious Service Call	157
	Honeybot Processors: Access Policy Processor: Incidents - Service Directory Indexing	157
	Honeybot Processors: Access Policy Processor: Incidents - Service Directory Spider	158
	Honeybot Processors: AJAX Processor	159
	Honeybot Processors: AJAX Processor: Incidents - Malicious Script Execution	160
	Honeybot Processors: AJAX Processor: Incidents - Malicious Script Introspection	161
	Honeybot Processors: Basic Authentication Processor	162
	Honeybot Processors: Basic Authentication Processor: Incidents - Apache Configuration Requested	163
	Honeybot Processors: Basic Authentication Processor: Incidents - Apache Password File Requested	164
	Honeybot Processors: Basic Authentication Processor: Incidents - Invalid Credentials	165
	Honeybot Processors: Basic Authentication Processor: Incidents - Protected Resource Requested	166
	Honeybot Processors: Basic Authentication Processor: Incidents - Password Cracked	166
	Honeybot Processors: Basic Authentication Processor: Incidents - Basic Authentication Brute Force	167
	Honeybot Processors: Cookie Processor	168
	Honeybot Processors: Cookie Processor: Incident - Cookie Parameter Manipulation	169
	Honeybot Processors: File Processor	170
	Honeybot Processors: File Processor: Incident - Suspicious Filename	170
	Honeybot Processors: File Processor: Incident - Suspicious File Exposed	171
	Honeybot Processors: File Processor: Incident - Suspicious Resource Enumeration	172
	Honeybot Processors: Hidden Input Form Processor	173
	Honeybot Processors: Hidden Input Form Processor: Incident - Parameter Type Manipulation	173
	Honeybot Processors: Hidden Input Form Processor: Incident - Hidden Parameter Manipulation	174
	Honeybot Processors: Hidden Link Processor	175
	Honeybot Processors: Hidden Link Processor: Incident - Link Directory Indexing	176
	Honeybot Processors: Hidden Link Processor: Incident - Link Directory Spidering	176
	Honeybot Processors: Hidden Link Processor: Incident - Malicious Resource Request	177
	Honeybot Processors: Query String Processor	177
	Honeybot Processors: Query String Processor: Incident - Query Parameter Manipulation	178
	Honeybot Processors: Robots Processor	179

Chapter 30

HoneyPot Processors: Robot Processor: Incident - Malicious Spider Activity . . .	179
Activity Processors	181
Activity Processors	182
Activity Processors: Custom Authentication Processor: Incident - Auth Input Parameter Tampering	183
Activity Processors: Custom Authentication Processor: Incident - Auth Query Parameter Tampering	184
Activity Processors: Custom Authentication Processor: Incident - Auth Cookie Tampering	184
Activity Processors: Custom Authentication Processor: Incident - Authentication Brute Force	185
Activity Processors: Custom Authentication Processor: Incident - Auth Invalid Login	185
Activity Processors: Cookie Protection Processor	186
Activity Processors: Cookie Protection Processor: Incident - Application Cookie Manipulation	187
Activity Processors: Error Processor	187
Activity Processors: Error Processor: Incident - Illegal Response Status	192
Activity Processors: Error Processor: Incident - Suspicious Response Status . . .	193
Activity Processors: Error Processor: Incident - Unexpected Response Status . .	193
Activity Processors: Error Processor: Incident - Unknown Common Directory Requested	194
Activity Processors: Error Processor: Incident - Unknown User Directory Requested	194
Activity Processors: Error Processor: Incident - Common Directory Enumeration	195
Activity Processors: Error Processor: Incident - User Directory Enumeration . . .	195
Activity Processors: Error Processor: Incident - Resource Enumeration	196
Activity Processors: Header Processor	197
Activity Processors: Header Processor: Incident - Duplicate Request Header . . .	198
Activity Processors: Header Processor: Incident - Duplicate Response Header	199
Activity Processors: Header Processor: Incident - Illegal Request Header	199
Activity Processors: Header Processor: Incident - Illegal Response Header . . .	200
Activity Processors: Header Processor: Incident - Missing All Headers	200
Activity Processors: Header Processor: Incident - Missing Host Header	200
Activity Processors: Header Processor: Incident - Missing Request Header . . .	201
Activity Processors: Header Processor: Incident - Missing Response Header . . .	201
Activity Processors: Header Processor: Incident - Missing User Agent Header . .	202
Activity Processors: Header Processor: Incident - Request Header Overflow . . .	202
Activity Processors: Header Processor: Incident - Unexpected Request Header	203
Activity Processors: Method Processor	203
Activity Processors: Method Processor: Incident - Illegal Method Requested . .	204
Activity Processors: Method Processor: Incident - Unexpected Method Requested	205
Activity Processors: Method Processor: Incident - Missing HTTP Protocol	205
Activity Processors: Method Processor: Incident - Unknown HTTP Protocol . . .	206

Chapter 31	Tracking Processors	207
	Tracking Processors: Etag Beacon Processor	207
	Tracking Processors: Etag Beacon Processor: Incident - Session Etag Spoofing	208
	Tracking Processors: Client Beacon Processor	209
	Tracking Processors: Client Beacon Processor: Incident - Beacon Parameter Tampering	210
	Tracking Processors: Client Beacon Processor: Incident - Beacon Session Tampering	211
	Tracking Processors: Client Fingerprint Processor	211
	Tracking Processors: Client Fingerprint Processor: Incident - Fingerprint Directory Indexing	214
	Tracking Processors: Client Fingerprint Processor: Incident - Fingerprint Directory Probing	214
	Tracking Processors: Client Fingerprint Processor: Incident - Fingerprint Manipulation	215
	Tracking Processors: Client Classification Processor	215
Chapter 32	Response Processors	219
	Response Processors	221
	Response Processors: Block Processor	222
	Response Processors: Request Captcha Processor	223
	Response Processors: Request Captcha Processor: Incident - Captcha Answer Automation	226
	Response Processors: Request Captcha Processor: Incident - No Captcha Answer Provided	227
	Response Processors: Request Captcha Processor: Incident - Multiple Captcha Request Overflow	228
	Response Processors: Request Captcha Processor: Incident - Unsupported Audio Captcha Requested	228
	Response Processors: Request Captcha Processor: Incident - Bad Captcha Answer	229
	Response Processors: Request Captcha Processor: Incident - Mismatched Captcha Session	230
	Response Processors: Request Captcha Processor: Incident - Expired Captcha Request	230
	Response Processors: Request Captcha Processor: Incident - Captcha Request Tampering	231
	Response Processors: Request Captcha Processor: Incident - Captcha Signature Tampering	232
	Response Processors: Request Captcha Processor: Incident - Captcha Signature Spoofing	233
	Response Processors: Request Captcha Processor: Incident - Captcha Cookie Manipulation	233
	Response Processors: Request Captcha Processor: Incident - Captcha Image Probing	234
	Response Processors: Request Captcha Processor: Incident - Captcha Request Size Limit Exceeded	235

Response Processors: Request Captcha Processor: Incident - Captcha Disallowed MultiPart	236
Response Processors: Request Captcha Processor: Incident - Captcha Directory Indexing	236
Response Processors: Request Captcha Processor: Incident - Captcha Directory Probing	237
Response Processors: Request Captcha Processor: Incident - Captcha Parameter Manipulation	238
Response Processors: Request Captcha Processor: Incident - Captcha Request Replay Attack	239
Response Processors: Request Captcha Processor: Incident - Multiple Captcha Replays	240
Response Processors: Request Captcha Processor: Incident - Multiple Captcha Disallow Multipart	241
Response Processors: Request Captcha Processor: Incident - Multiple Captcha Parameter Manipulation	242
Response Processors: CSRF Processor	243
Response Processors: CSRF Processor: Incident - CSRF Parameter Tampering	245
Response Processors: CSRF Processor: Incident - Multiple CSRF Parameter Tampering	246
Response Processors: CSRF Processor: Incident - CSRF Remote Script Inclusion	247
Response Processors: CSRF Processor: Incident - HTTP Referers Disabled	247
Response Processors: Header Injection Processor	248
Response Processors: Force Logout Processor	248
Response Processors: Strip Inputs Processor	249
Response Processors: Slow Connection Processor	249
Response Processors: Warning Processor	250
Response Processors: Warning Processor: Incident - Warning Code Tampering	251
Response Processors: Application Vulnerability Processor	252
Response Processors: Application Vulnerability Processor: Incident - App Vulnerability Detected	252
Response Processors: Support Processor	253
Response Processors: Cloppy Processor	254
Response Processors: Login Processor	255
Response Processors: Login Processor: Incident - Site Invalid Login	261
Response Processors: Login Processor: Incident - Site Login Multiple IP	262
Response Processors: Login Processor: Incident - Site Login Multiple Usernames	262
Response Processors: Login Processor: Incident - Site Login User Sharing	263
Response Processors: Login Processor: Incident - Site Login User Pooling	263
Response Processors: Login Processor: Incident - Site Login User Brute Force	264
Response Processors: Login Processor: Incident - Site Login Brute Force	264
Response Processors: Login Processor: Incident - Site Login Username Scan	264
Response Processors: Google Map Processor	265

Chapter 33	Incident Methods	267
	List Of Incident Methods	267
Chapter 34	Captcha Template	271
	Captcha Template	271
Chapter 35	Log Format	275
	Log Format	275
	Incident Log Format	275
	Counter Response Log Format	276
	Profile Log Format	277
Part 5	Index	
	Index	281

List of Figures

Part 1	Overview	
Chapter 2	Deployment	7
	Figure 1: WebApp Secure Placement in the Network - Between Firewall and Web Servers	8
	Figure 2: WebApp Secure Deployment - Connected to Load Balancer	9
Chapter 3	Configuration	11
	Figure 3: Configuration, Main Screen	13
Chapter 10	Reports	27
	Figure 4: Reporting Interface	27
	Figure 5: Scheduled Reports	28
Part 2	Configuration	
Chapter 11	Initial Tasks	31
	Figure 6: TUI, Initialize Appliance	32
	Figure 7: TUI, Set Hostname	32
	Figure 8: TUI, Enter Hostname	33
	Figure 9: TUI, Configure Interface	33
	Figure 10: TUI, Network Interfaces	34
	Figure 11: TUI, Configure Interface	34
	Figure 12: TUI, Set DNS	34
	Figure 13: TUI, Restart Network	35
	Figure 14: TUI, Initialize Appliance	35
	Figure 15: TUI, Initialize System	36
	Figure 16: Appliance Login Screen	37
	Figure 17: License Terms	38
Chapter 12	Configuration Wizard	39
	Figure 18: Wizard, Configure Backend Servers, Step 1	39
	Figure 19: Wizard, Configure Backend Servers, Step 2	40
	Figure 20: Wizard, Configure SMTP Settings, Step 3	41
	Figure 21: Wizard, Configure Alert Service, Step 4	41
	Figure 22: Wizard, Configure Alert Service, Step 5	42
	Figure 23: Wizard, Configure Alert Service, SNMP, Step 6	42
	Figure 24: Wizard, Configure Alert Service, Email Contacts, Step 7	43
	Figure 25: Wizard, Configure Backup Service	43
	Figure 26: Wizard, Configure Spotlight Secure	44
	Figure 27: Wizard, Confirmation Page	45
Chapter 14	Configuration Options	51

	Figure 28: Configured Applications	52
	Figure 29: Application Wizard	52
	Figure 30: Application Dashboard	53
	Figure 31: URL Pattern	54
	Figure 32: Application Patterns	54
	Figure 33: Define Backend Servers	55
	Figure 34: Add New Page	56
	Figure 35: Proxy / Backends	57
	Figure 36: Whitelists	58
Chapter 15	Clustering	61
	Figure 37: TUI, Initialize Appliance	62
	Figure 38: TUI, Select System Mode	62
	Figure 39: TUI, Initialization Complete	62
	Figure 40: TUI Select Traffic Processor	63
	Figure 41: TUI, Master Node IP Address	63
Chapter 16	High Availability	65
	Figure 42: TUI, Select HA	66
	Figure 43: TUI, Enter HA Node Addresses	67
	Figure 44: TUI, Enter Virtual Addresses	67
	Figure 45: HA Pair Status	68
Chapter 17	SRX Integration	71
	Figure 46: Initialize Filter	72
	Figure 47: Create Filter Term	72
	Figure 48: Bind Filter to Interface	73
Chapter 18	Reports	77
	Figure 49: Schedule Report - Scorecard	78
	Figure 50: Schedule Report - Country Comparison Over Time	79
	Figure 51: Reports CLI	80
Part 3	Administration	
Chapter 19	General Tasks	85
	Figure 52: Boot Menu	85
	Figure 53: Reset Password	86
Chapter 21	EC2 Deployment	89
	Figure 54: AWS management console	90
	Figure 55: Instance Type	91
	Figure 56: Configure Instance continued	91
	Figure 57: Instance, Configure Name	92
	Figure 58: Instance, Create Key Pair	92
	Figure 59: Instance Create Security Group	93
	Figure 60: Instance, Review and Launch	94
	Figure 61: Allocate New Address	95
Chapter 22	Configuration Modes and Roles	97
	Figure 62: Edit Parameter	98
	Figure 63: Users and Groups, Add User	100

	Figure 64: Assigned Roles	100
	Figure 65: User Preferences	104
	Figure 66: Blocked Login	104
Chapter 24	Appliance Management	109
	Figure 67: Dashboard, Updates	109
	Figure 68: Downloading Update	110
	Figure 69: Update Description	111
	Figure 70: Updating the Application	111
	Figure 71: CPU Utilization	112
	Figure 72: CPU Load Average	113
	Figure 73: Memory Utilization	113
	Figure 74: Network Traffic	114
	Figure 75: Proxy Connections	114
	Figure 76: Proxy Requests	115
	Figure 77: Backup Configuration	116
	Figure 78: Backup Configuration	117
	Figure 79: Restore Backup	118
Chapter 25	Spotlight Secure	119
	Figure 80: Spotlight Secure, Enable	119
	Figure 81: Spotlight Secure Configuration Screen	120
	Figure 82: Recent Attackers: Global and Local Names	120
	Figure 83: Recent Attackers: Global Names	121
	Figure 84: User Preferences: Select Spotlight Name Preference	122
Chapter 26	Security Monitor	123
	Figure 85: Security Monitor Dashboard	124
	Figure 86: Dashboard - Filter By tab	124
	Figure 87: User Preferences	125
	Figure 88: Recent Attackers	129
	Figure 89: Attacker Profile	130
	Figure 90: Responses tab - Deactivate	131
	Figure 91: System Status	140
	Figure 92: Services Status	140
	Figure 93: Updates	141
Chapter 27	Autoresponse Defaults and Rule Creation	143
	Figure 94: Autoresponse	143

List of Tables

	About the Documentation	xix
	Table 1: Notice Icons	xx
	Table 2: Text and Syntax Conventions	xx
Part 2	Configuration	
Chapter 14	Configuration Options	51
	Table 3: Luna Control Center Configuration Changes	58
	Table 4: WebApp Secure Configuration Settings for Akamai Support	59
Chapter 17	SRX Integration	71
	Table 5: External Counter Response Service Configuration Parameters	74
Part 3	Administration	
Chapter 22	Configuration Modes and Roles	97
	Table 6: RBAC Groups and Roles.	101
Chapter 24	Appliance Management	109
	Table 7: Health Check responses and corresponding meanings.	116
Chapter 26	Security Monitor	123
	Table 8: Security Monitor Dashboard Panes	126
Chapter 27	Autoresponse Defaults and Rule Creation	143
	Table 9: Autoresponse Descriptions	145
	Table 10: Autoresponse Editor Fields	147
Part 4	Monitoring	
Chapter 29	Honeypot Processors	155
	Table 11: Access Policy Processor Configuration Parameters	156
	Table 12: AJAX Processor Configuration Parameters	159
	Table 13: Basic Authentication Processor Configuration Parameters	162
	Table 14: Cookie Processor Configuration Parameters	169
	Table 15: File Processor Configuration Parameters	170
	Table 16: Hidden Input Form Processor Configuration Parameters	173
	Table 17: Hidden Link Processor Configuration Parameters	175
	Table 18: Query String Processor Configuration Parameters Parameter Type Default Value Description	177
	Table 19: Robots Processor Configuration Parameters	179
Chapter 30	Activity Processors	181

	Table 20: Custom Authentication Processor Configuration Parameters	182
	Table 21: Cookie Protection Processor Configuration Parameters	186
	Table 22: Error Processor Configuration Parameters	188
	Table 23: Header Processor Configuration Parameters	197
	Table 24: Method Processor Configuration Parameters	204
Chapter 31	Tracking Processors	207
	Table 25: Etag Beacon Processor Configuration Parameters	207
	Table 26: Client Beacon Processor Configuration Parameters	209
	Table 27: Client Fingerprint Configuration Parameters	211
	Table 28: Client Classification Configuration Parameters	216
Chapter 32	Response Processors	219
	Table 29: Block Processor Configuration Parameters	222
	Table 30: Request Captcha Processor Configuration Parameters	223
	Table 31: CSRF Processor Configuration Parameters	244
	Table 32: Header Injection Processor Configuration Parameters	248
	Table 33: Force Logout Processor Configuration Parameters	249
	Table 34: Strip Inputs Processor Configuration Parameters	249
	Table 35: Slow Connection Processor Configuration Parameters	250
	Table 36: Warning Processor Configuration Parameters	250
	Table 37: Application Vulnerability Processor Configuration Parameters	252
	Table 38: Support Processor Configuration Parameters	254
	Table 39: Cloppy Processor Configuration Parameters	255
	Table 40: Login Processor Configuration Parameters	258
	Table 41: Google Map Processor Configuration Parameters	265
Chapter 33	Incident Methods	267
	Table 42: Incident Methods	267

About the Documentation

- Documentation and Release Notes on page xix
- Documentation Conventions on page xix
- Documentation Feedback on page xxi
- Requesting Technical Support on page xxi

Documentation and Release Notes

To obtain the most current version of all Juniper Networks® technical documentation, see the product documentation page on the Juniper Networks website at <http://www.juniper.net/techpubs/>.

If the information in the latest release notes differs from the information in the documentation, follow the product Release Notes.

Juniper Networks Books publishes books by Juniper Networks engineers and subject matter experts. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration. The current list can be viewed at <http://www.juniper.net/books>.

Documentation Conventions

Table 1 on page xx defines notice icons used in this guide.

Table 1: Notice Icons

Icon	Meaning	Description
	Informational note	Indicates important features or instructions.
	Caution	Indicates a situation that might result in loss of data or hardware damage.
	Warning	Alerts you to the risk of personal injury or death.
	Laser warning	Alerts you to the risk of personal injury from a laser.

Table 2 on page xx defines the text and syntax conventions used in this guide.

Table 2: Text and Syntax Conventions

Convention	Description	Examples
Bold text like this	Represents text that you type.	To enter configuration mode, type the configure command: <code>user@host> configure</code>
<code>Fixed-width text like this</code>	Represents output that appears on the terminal screen.	<code>user@host> show chassis alarms</code> <code>No alarms currently active</code>
<i>Italic text like this</i>	<ul style="list-style-type: none"> Introduces or emphasizes important new terms. Identifies guide names. Identifies RFC and Internet draft titles. 	<ul style="list-style-type: none"> A policy <i>term</i> is a named structure that defines match conditions and actions. <i>Junos OS CLI User Guide</i> RFC 1997, <i>BGP Communities Attribute</i>
<i>Italic text like this</i>	Represents variables (options for which you substitute a value) in commands or configuration statements.	Configure the machine's domain name: <code>[edit]</code> <code>root@# set system domain-name <i>domain-name</i></code>
Text like this	Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components.	<ul style="list-style-type: none"> To configure a stub area, include the stub statement at the <code>[edit protocols ospf area area-id]</code> hierarchy level. The console port is labeled CONSOLE.
< > (angle brackets)	Encloses optional keywords or variables.	<code>stub <default-metric <i>metric</i>>;</code>

Table 2: Text and Syntax Conventions (*continued*)

Convention	Description	Examples
(pipe symbol)	Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity.	broadcast multicast <i>(string1 string2 string3)</i>
# (pound sign)	Indicates a comment specified on the same line as the configuration statement to which it applies.	rsvp { # Required for dynamic MPLS only
[] (square brackets)	Encloses a variable for which you can substitute one or more values.	community name members [community-ids]
Indentation and braces ({ })	Identifies a level in the configuration hierarchy.	[edit] routing-options { static { route default { nexthop address; retain; } } }
;(semicolon)	Identifies a leaf statement at a configuration hierarchy level.	
GUI Conventions		
Bold text like this	Represents graphical user interface (GUI) items you click or select.	<ul style="list-style-type: none"> In the Logical Interfaces box, select All Interfaces. To cancel the configuration, click Cancel.
> (bold right angle bracket)	Separates levels in a hierarchy of menu selections.	In the configuration editor hierarchy, select Protocols>Ospf .

Documentation Feedback

We encourage you to provide feedback, comments, and suggestions so that we can improve the documentation. You can send your comments to techpubs-comments@juniper.net, or fill out the documentation feedback form at <https://www.juniper.net/cgi-bin/docbugreport/>. If you are using e-mail, be sure to include the following information with your comments:

- Document or topic name
- URL or page number
- Software release version (if applicable)

Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active J-Care or JNASC support contract,

or are covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

- JTAC policies—For a complete understanding of our JTAC procedures and policies, review the *JTAC User Guide* located at <http://www.juniper.net/us/en/local/pdf/resource-guides/7100059-en.pdf>.
- Product warranties—For product warranty information, visit <http://www.juniper.net/support/warranty/>.
- JTAC hours of operation—The JTAC centers have resources available 24 hours a day, 7 days a week, 365 days a year.

Self-Help Online Tools and Resources

For quick and easy problem resolution, Juniper Networks has designed an online self-service portal called the Customer Support Center (CSC) that provides you with the following features:

- Find CSC offerings: <http://www.juniper.net/customers/support/>
- Search for known bugs: <http://www2.juniper.net/kb/>
- Find product documentation: <http://www.juniper.net/techpubs/>
- Find solutions and answer questions using our Knowledge Base: <http://kb.juniper.net/>
- Download the latest versions of software and review release notes: <http://www.juniper.net/customers/csc/software/>
- Search technical bulletins for relevant hardware and software notifications: <https://www.juniper.net/alerts/>
- Join and participate in the Juniper Networks Community Forum: <http://www.juniper.net/company/communities/>
- Open a case online in the CSC Case Management tool: <http://www.juniper.net/cm/>

To verify service entitlement by product serial number, use our Serial Number Entitlement (SNE) Tool: <https://tools.juniper.net/SerialNumberEntitlementSearch/>

Opening a Case with JTAC

You can open a case with JTAC on the Web or by telephone.

- Use the Case Management tool in the CSC at <http://www.juniper.net/cm/>.
- Call 1-888-314-JTAC (1-888-314-5822 toll-free in the USA, Canada, and Mexico).

For international or direct-dial options in countries without toll-free numbers, see <http://www.juniper.net/support/requesting-support.html>.

PART 1

Overview

- [WebApp Secure on page 3](#)
- [Deployment on page 7](#)
- [Configuration on page 11](#)
- [Clustering on page 15](#)
- [High Availability on page 17](#)
- [SRX Integration on page 19](#)
- [Appliance Management on page 21](#)
- [Spotlight Secure on page 23](#)
- [The Processors on page 25](#)
- [Reports on page 27](#)

CHAPTER 1

WebApp Secure

- [WebApp Secure Summary on page 3](#)
- [How It Works on page 3](#)
- [Four Phases of Detection and Response on page 4](#)
- [WebApp Secure Features on page 4](#)
- [WebApp Secure Services on page 5](#)
- [Limitations on page 6](#)
- [WebApp Secure Appliance Terminology on page 6](#)
- [Using WebApp Secure with Third-Party Load Balancer on page 6](#)

WebApp Secure Summary

WebApp Secure protects web sites from would-be attackers, fraud and theft. Its web intrusion prevention system uses deception to detect, track, profile and block attackers in real-time by inserting detection points into your web server's output to identify attackers before they do damage. WebApp Secure then tracks detected attackers, profiling their behavior and deploying counter measures.

How It Works

WebApp Secure sits between your web servers and the outside world. It inspects HTTP and HTTPS traffic and functions as a reverse proxy. WebApp Secure seeks out potential attack attempts or probes by adding detection points to outbound web traffic and removing detection points from inbound web traffic. These detection points are transparent to common, legitimate users. It then monitors and strips these points from the requests coming back from the user's browser. Any change to a detection point is an indicator of an attempted attack. The system logs incidents to a database of attacker profiles, and exposes them to the security administrators through a web-based interface. System administrators can then apply automated abuse-prevention policies, or respond manually.

Four Phases of Detection and Response

WebApp Secure protects your web servers by doing the following:

1. **Detect:** Use traps to detect attackers before they are able to formulate attacks. Detection points discover attackers doing reconnaissance on your site, looking for vulnerabilities to exploit.
2. **Trap:** Persistent tokens and fingerprinting track IP addresses, browsers, software, and scripts.
3. **Profile:** Create a smart profile of threats, helping you to understand the threat and determine the best way to respond.
4. **Respond:** After detecting, tracking, and profiling the threat, you have many choices for responding to the attacker. You can slow his connection causing him to waste valuable time, warn him that he is being watched, or completely block him from your application.

WebApp Secure Features

Below are some of the highlights of the features included available with WebApp Secure:

- **Abuse Profiles Tracking** - Maintain a historical profile of known application abusers and all of their malicious activity against the application for analysis and sharing.
- **Counter Responses** - Respond to application abuse with session-specific deceptive responses, warnings, and blocks.
- **HTTP Capture** - Capture and log HTTP traffic for security incidents.
- **Multi-application Protection** - Secure traffic for multiple application domains.
- **Intrusion Deception based protection** - Enable specific abuse detection points in application code using a library of security modules (referred to as Processors).
- **Tagging and Re-identification** - Re-identify abusive users and apply persistent responses over time and across sessions.
- **SSL Inspection** - Passive decryption or termination.
- **Web User Interface** - Examine application sessions, security incidents, and abuse profiles using the Web-based monitoring and administration interface. Manage and monitor responses and configure the system.
- **Software and Hardware Delivery Support** - Distributed as VMWare image, hardware appliance or Amazon AMI image.
- **Alerting** - Configure WebApp Secure to send alerts to administrators security incidents via SNMP or SMTP.
- **Spotlight Secure Connectivity** - Spotlight Secure is a global service designed to provide additional intelligence in threat detection and prevention. If enabled, it will share

information about attackers and attacks gathered across multiple customers and adds yet another layer of intelligence to WebApp Secure attacker tracking.

- **Authentication and Authorization** – Connect your WebApp Secure instance to LDAP or RADIUS to use common authentication across your datacenter. Assign roles to specific users for more granular access controls.

WebApp Secure Services

WebApp Secure includes the following services which run in the background:

- **Alert Service** Configure WebApp Secure to send email alerts to administrators when an incident of a specified severity is detected. For instance, the system can send out an email notification to a specific administrator who is on call if an incident level of critical is detected, allowing the administrator to respond quickly to the threat. It can also send alerts to email addresses on a defined schedule, and/or send SNMP traps to one or more SNMP servers. The initial configuration of the Alert Service can be performed with the Configuration Wizard, but if you need to change these settings, they are available through the "Services" tab of Configuration.
- **Auto Response Service** Turn the Auto Response service on or off.
- **Backup Service** Configure the frequency, type, and destination for system backups. These settings can also be configured through the Configuration Wizard.
- **Database Cleanup Service** Configure how often the database cleanup service runs for sessions, profiles, and incidents through this option.
- **Security Engine Service** Configure the memory, database, and fingerprinting used by the security engine.
- **Session Consolidation Service** Configure how and when the system consolidates user sessions.
- **Statistical Service** Configure how the system logs statistical data.
- **Spotlight Secure** Spotlight Secure is designed to provide additional intelligence. If enabled, it will share information about attackers and attacks it is observing using the Spotlight server run by Juniper Networks. Juniper will also consume information from the Spotlight service. This allows customers to identify attackers (that have attacked other Juniper customers). This service also provides additional details about sessions which allows Juniper to make more informed decisions on how to respond to threats. By default, the service is turned off. If you would like more information about the Spotlight service, please contact Juniper Networks.

Limitations

WebApp Secure has the following limitations:

- WebApp Secure is an L7 reverse web proxy and does not support hardware fail open capability itself; it should not be physically in line with protected application servers.
- WebApp Secure only accepts HTTP and HTTPS 1.0 and 1.1 traffic because it is the only traffic it monitors.
- WebApp Secure is not a network firewall and should not be an edge device. While firewall capabilities exist on the device, they are not available to the user and are there to protect the WebApp Secure device itself.
- WebApp Secure does not support NTLM authentication.

WebApp Secure Appliance Terminology

The following terminology is used in this section:

- **Appliance:** The software/hardware system. It is synonymous with WebApp Secure in most contexts.
- **backend:** In almost all cases (except where explicitly mentioned otherwise) the backend is defined as the server which houses the web application that is being protected by WebApp Secure.
- **Application:** The protected web application. It can also reference the web server program, whether it is Apache, JBOSS, Microsoft, or other web serving software.
- **GUI:** Graphical User Interface. In cases referencing the WebApp Secure system, this term refers to the Web Interface.
- **HA:** High Availability, a configuration that aims to reduce the chance full system failures.
- **TUI:** Text User Interface, usually invoked from the command line using **sudo setup**.

Using WebApp Secure with Third-Party Load Balancer

If you are using the appliance with a third-party load balancer, you must make sure to tell WebApp Secure to accept the X-Forwarded-For header from the load balancer. If this is not set, all IPs in the appliance will seem to be coming from the load balancer directly.

To trust the X-Forwarded-For header, SSH into the appliance, and enter the following:

```
sudo mykonos-shell config set engine.exclude_forward_addresses  
<IP_of_Loadbalancer>
```

This tells WebApp Secure to trust the header of your load balancer.

CHAPTER 2

Deployment

- [Appliance Deployment Overview on page 7](#)
- [Placement Between Firewall and Web Servers on page 7](#)
- [Options for Load-Balanced Environments on page 9](#)
- [SSL Traffic Considerations on page 9](#)
- [EC2 Deployment on page 10](#)
- [Using a Secure Cluster on page 10](#)

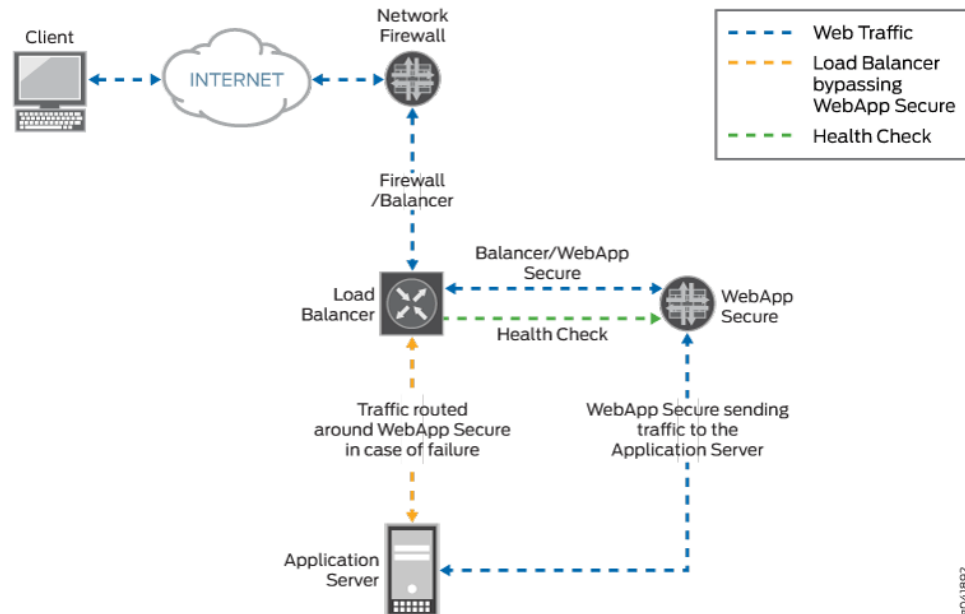
Appliance Deployment Overview

The WebApp Secure appliance processes all inbound web requests and outbound web responses. Outbound responses are modified in ways that are invisible to the average user; inbound requests are checked to see if the modified responses have been altered in any way. Any alterations are suspicious and indicate a possible hacker. Due to its focus on web applications, WebApp Secure only accepts HTTP/HTTPS traffic and is normally placed between a load balancer and your web applications. Topologically, you should think of WebApp Secure as a web reverse proxy server.

Placement Between Firewall and Web Servers

WebApp Secure acts as a reverse proxy and actively manipulates traffic between the protected web application and the Internet. It is deployed between the protected web server and the last system which can alter user-facing traffic. This location gives WebApp Secure full visibility into the HTTP traffic destined for the web servers (including any errors caused by authentication failures), and lets it inject and strip out any code it uses in protecting the application. This topology has the added benefit of minimally impacting internal network bandwidth. The following figure shows the WebApp Secure deployed in its most simple form as a reverse proxy connected to a load balancer.

Figure 1: WebApp Secure Placement in the Network - Between Firewall and Web Servers

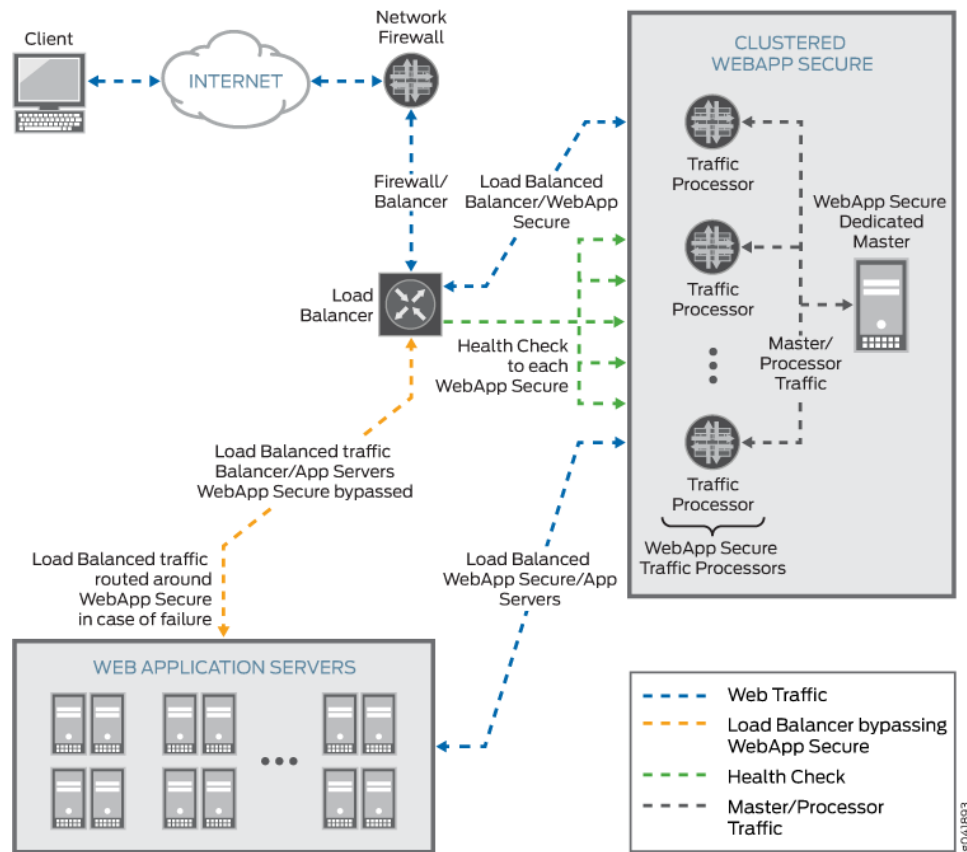


Network placement requirements for WebApp Secure are as follows:

- Because WebApp Secure only processes HTTP and HTTPS traffic, it must live behind a device that can separate Application Layer (Layer 7) traffic.
- In order to prevent a WebApp Secure issue from impacting a protected application, the upstream device (i.e., the router or load balancer) must perform Health Check monitoring on WebApp Secure over HTTP. If the Health Check fails, the load balancer or Layer 7 router should pass traffic directly to the protected application servers, rather than to WebApp Secure.

The actual implementation depends on the user's specific network topology. The following figure shows a more complex environment with clustered web servers and clustered appliances.

Figure 2: WebApp Secure Deployment - Connected to Load Balancer



Options for Load-Balanced Environments

WebApp Secure can serve as a load balancer for HTTP and HTTPS web traffic, but it is recommended that a dedicated hardware solution be used in that capacity. Dedicated load balancers are optimized for that role and will provide higher overall performance.

SSL Traffic Considerations

WebApp Secure includes SSL decryption capabilities to give it visibility into all of the protected application's traffic. It supports two modes: Passive Decryption and SSL Termination. In Passive Decryption mode, WebApp Secure decrypts requests for processing, then re-encrypts them before sending them on to the application server. HTTPS responses to the user follow the same process, where they are decrypted, processed, and re-encrypted before returning to the user. In SSL Termination mode, the appliance serves as an SSL termination point. It decrypts incoming HTTPS traffic, processes them, then proxies the decrypted requests on to the application. Responses to the user are received unencrypted from the application server, processed, encrypted, then passed to the user.

EC2 Deployment

The WebApp Secure instance is a private instance. For access, you must provide Juniper your account id. You can find your account id when you log into your AWS account and select **Account Activity**. Your account id is displayed in the top right under your account name.

Using a Secure Cluster

Individual WebApp Secure appliances have the ability to work together as one system in a cluster. Clustering allows traffic to be divided among multiple appliances, effectively reducing the per-system load. In a clustered network configuration, the master node holds the database that is populated by one or more traffic processors. In order to successfully utilize a WebApp Secure cluster, a load-balancer must properly segregate traffic to each of the defined traffic processing nodes. Each of these traffic nodes must maintain connectivity with the master in order to operate.



.....

NOTE: Clustering should not be confused with High Availability. Clustering is used to increase throughput (by utilizing multiple processing nodes), and can reduce the chance that the whole system will fail. Clustering does not protect the master node from failure as in a High Availability setup; only HA configurations are set up to include failsafe procedures to designate a new master when the first one is unavailable.

.....

CHAPTER 3

Configuration

- [First Time Configuration on page 11](#)
- [Web Interface Configuration Overview on page 11](#)
- [About the Configuration Wizard on page 12](#)
- [Basic vs. Expert Configuration on page 12](#)
- [Using the CLI on page 13](#)

First Time Configuration

Basic configuration of WebApp Secure is a straightforward process executed using the console and the Text User Interface (TUI).

- Set the host name
- Set up networking TUI
- Restart networking
- Initialize the appliance

At this point, the web interface should be active. From the web interface , you will:

- License the appliance
- Run the installation wizard
- Perform a basic appliance test

Web Interface Configuration Overview

The Web interface is used for system configuration, as well as monitoring and reporting. The initial installation required you to access this interface to license WebApp Secure and to bring your appliance on-line. You will use this interface for nearly all configuration options.

The web interface URL is: **https://<IP address or hostname>:5000**

For example:

- **https://10.11.12.13:5000**

- <https://webappsecure.mydomain.com:5000>



NOTE: Log into the Web interface using the default account. If you have not changed the password, the default is `mykonosadmin`. It is strongly recommended that you change this password.

About the Configuration Wizard

The Configuration Wizard helps you configure the most commonly used features on the WebApp Secure appliance, including defining one or more web applications to protect, and setting up alerting and backups. Now that the appliance has network connectivity, it needs one or more Backend Servers (web applications) defined in order to protect live traffic.

The wizard walks you through the process of setting up Backend Servers, SMTP, Alerts, and Backups. Once you've completed the wizard, a confirmation page is displayed. It also provides some additional steps such as pointing your load balancer to the appliance.



WARNING: Upon completion of the wizard, make a note of your backup encryption key. If you lose this key, no one - including Juniper Support, can retrieve the information contained in your backups.



NOTE: When using the system's default mail server, set a valid hostname and ensure that the mail configuration observes all best practices for setting up a mail server.



NOTE: The wizard has a minimum of 6 steps. The actual number of steps may increase depending on your choices, such as defining multiple applications to protect.

Basic vs. Expert Configuration

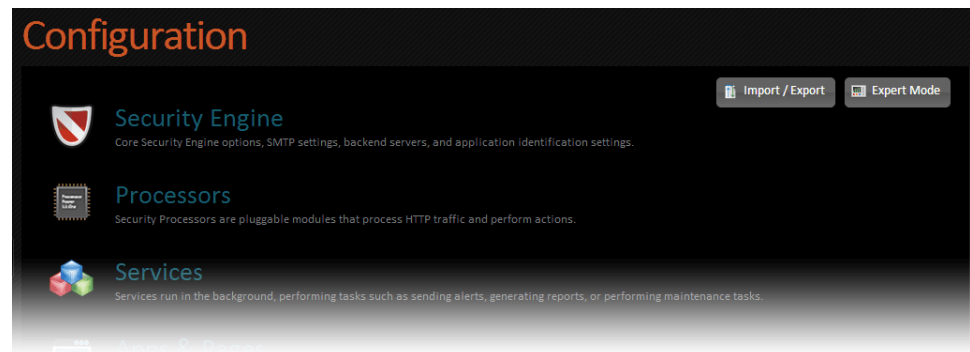
WebApp Secure is widely configurable with numerous settings to optimize it for your environment. Once you've completed the Wizard, the Appliance will route traffic through a default application profile to the designated backend servers. This default profile is simple, but adequate for many applications. However, there are still hundreds of options available to further customize the Appliance to meet your specific requirements. WebApp Secure has two configuration modes accessible from the Configuration button on the Web interface. Basic Mode gives you access to all of the Appliance's features with a wide range of customizations through a user friendly interface and is recommended for most situations. Expert mode, on the other hand, gives you access to the deepest levels of the Appliance's configuration presented in a key:value pair format. Expert mode is just that:

best used by experts who are comfortable making multiple changes at once. Basic mode is recommended for most users and most applications.



NOTE: When using Expert Mode, be sure to click the "Save" button when you are done making changes. Unlike Basic Mode, Expert Mode does NOT save the configuration to the engine after each parameter is modified; it lets you make multiple changes at once and then write the entire configuration image as one transaction.

Figure 3: Configuration, Main Screen



Using the CLI

While the Web-based configuration handles most configuration needs, the WebApp Secure CLI is also available for changing the appliance configuration. You can access the CLI via the shell, by entering **sudo mykonos-shell** from the command line.

There are currently three main systems within the CLI as follows:

- **config:** allows you to access the appliance configuration
- **services:** allows you to perform tasks on the various background services functioning on the appliance
- **system:** allows you to perform tasks on the various background services functioning on the appliance, and system allows you to shutdown or reboot the appliance



NOTE: At any point, you can enter **?** or **help** to view the available commands at the current context, or **help <command>** to get contextual help on the specified command. You can also show information on a particular parameter by using the **info** command. For example, **info services.cleanup.db.enabled**. Entering **Ctrl + D** or **exit** will leave the context you are currently in, or exit the CLI. The CLI also has full tab-complete support.

CHAPTER 4

Clustering

- [Using a Secure Cluster on page 15](#)
- [Node Types on page 15](#)

Using a Secure Cluster

Individual WebApp Secure appliances have the ability to work together as one system in a cluster. Clustering allows traffic to be divided among multiple appliances, effectively reducing the per-system load. In a clustered network configuration, the master node holds the database that is populated by one or more traffic processors. In order to successfully utilize a WebApp Secure cluster, a load-balancer must properly segregate traffic to each of the defined traffic processing nodes. Each of these traffic nodes must maintain connectivity with the master in order to operate.



NOTE: Clustering should not be confused with High Availability. Clustering is used to increase throughput (by utilizing multiple processing nodes), and can reduce the chance that the whole system will fail. Clustering does not protect the master node from failure as in a High Availability setup; only HA configurations are set up to include failsafe procedures to designate a new master when the first one is unavailable.

Node Types

In a traditional WebApp Secure deployment (one system), the appliance is responsible for holding its own database as well as processing the traffic. In a clustered deployment, you have the ability to segregate the database from those systems which will process incoming requests. During cluster configuration, you will have the ability to designate a node type for each system. At a minimum, the cluster must have a way to process traffic and a way to store the relevant information.

Node types are as follows:

- **Master:** A master node is similar to a single-system deployment in that it holds the database, and also processes incoming traffic. This satisfies both requirements for a cluster (database and traffic processor). It is possible to set up a cluster with only one

master node (no additional processing nodes). Additional traffic processing nodes can be added at a later point in time if desired.

- **Dedicated Master:** A dedicated master node holds the database similar to a master node, but it does not have the ability to process traffic. Using a dedicated master in a clustered configuration requires the addition of at least one traffic node.
- **Traffic Processor:** A traffic node is only responsible for processing incoming requests. It does not contain a database, so a master or a dedicated master node must accompany a traffic node. The number of traffic nodes you can add to a cluster is dependent on (1.) the hardware specifications of the master, (2.) the amount of incoming traffic on protected web application, and (3.) the number of additional traffic nodes in the cluster. For optimal stability, be sure to monitor the cluster's performance as you add each traffic node.

CHAPTER 5

High Availability

- [High Availability Overview on page 17](#)

High Availability Overview

To minimize the risk of downtime, WebApp Secure deployments have the ability to be placed in a Highly Available (HA) configuration. In this setup, an additional appliance is on stand-by in the event that the currently-active appliance goes offline. If this happens, the passive appliance is able to become the new active appliance automatically - without needing to restart the system. An HA configuration is similar to clustering, with the major exception being that the passive system has a copy of the services needed to take over when the master fails. WebApp Secure uses a Virtual IP (VIP) to float between the currently active system and the current passive system.



NOTE: An HA configuration is only available on WebApp Secure dedicated hardware systems. It is not available in a Virtual Machine installation.

CHAPTER 6

SRX Integration

- [Integration with SRX Overview on page 19](#)

Integration with SRX Overview

The SRX series by Juniper is an enterprise-level Secure Gateway for networks. WebApp Secure has the ability to integrate with this solution, which means it can send IPs to the SRX to achieve a block (or other configurable response) at the gateway level. This effectively allows the SRX to tap into the identifying metrics produced by WebApp Secure.

CHAPTER 7

Appliance Management

- [Appliance Management Overview on page 21](#)

Appliance Management Overview

A few management tasks are organized into the Text User Interface, or TUI for short. You can reach the TUI via SSH to your appliance on port 2022. Normal console login will take the administrator directly to the TUI, but if you are at a shell prompt you can start the TUI with the command: `sudo setup`. From here you can initialize the appliance (essentially reformatting it to the state it was shipped), configure network components of the appliance, and send a set of information to support staff.

CHAPTER 8

Spotlight Secure

- [About Spotlight Secure on page 23](#)

About Spotlight Secure

Spotlight Secure -- or simply Spotlight provides a database of known attackers to WebApp Secure for use throughout the appliance. This two-way link enables WebApp Secure to block attackers based only on a unique and specialized fingerprint gathered by a completely different WebApp Secure installation. It also provides a mechanism for reporting attacker information gathered on the local installation to the Global Attacker Database. Because your local WebApp Secure appliance is relaying information to a central data store, the ability to recognize attacker quickly and effectively increases as the database grows.

Here is an overview of how Spotlight Secure works:

1. A user gets profiled by WebApp Secure.
2. WebApp Secure sends a unique client fingerprint that is unique to that user.
3. The Spotlight service searches its Global Attacker Database for an attacker with the same fingerprint.
4. If a match is found, Spotlight feeds all identifying information on that user to the WebApp Secure appliance automatically.
5. If the user is not doing anything malicious, and is not found currently within Spotlight's database, the fingerprint for the user is still stored within the local session.
6. If at any point the user becomes malicious and is flagged by WebApp Secure, the appliance will submit the fingerprint and other data to the Spotlight service for inclusion in the Global Attacker Database.

CHAPTER 9

The Processors

- [Processors Overview on page 25](#)

Processors Overview

WebApp Secure uses a modular approach to securing your application. Each module is responsible for monitoring, detecting and securing a particular aspect of the application and/or individual HTTP request/ response. These logical entities are referred to as Security Processors. Processors are the configurable operators that implement an additional layer of security between the application/web servers and the end user. They are responsible for analyzing the request and response data sent to and from the server and they monitor anything from the state of injected honey pots to contents of the headers and the body of the HTTP/HTTPS requests and responses.

Processors can be managed through the system configuration user interface. While some of the operations may be as simple as incrementing a counter, others are far more sophisticated and may alter the request and response data so it is important that you configure processors correctly to ensure web application's security and functionality.

Each processor is monitoring the HTTP stream for particular alterations from what is considered typical traffic. These alterations are called "triggers". Each security processor may have several triggers they are responsible for detecting. If matched, the processor responsible for handling it will generate a security incident. Incident varies by its complexity, which is explained in the section below.

CHAPTER 10

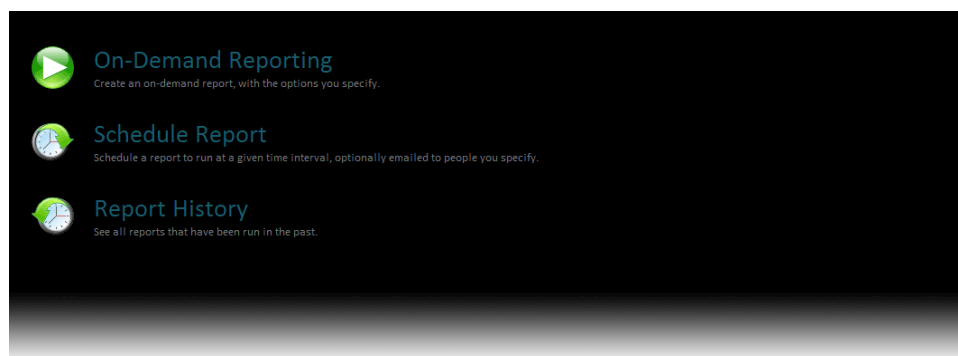
Reports

- [Reporting Overview on page 27](#)
- [Scheduling a Report Overview on page 27](#)

Reporting Overview

WebApp Secure has a built-in reporting interface that can be accessed through the Web UI, by navigating to the **Reports** menu item on the left-hand side. Administrators can run one of many pre-defined reports or schedule a report or access report history. Most reports can be exported to both PDF and CSV (comma separated value) formats. Reports that are composed of several disparate visual elements (like the Scorecard) are only available in PDF format.









Figure 4: Reporting Interface



Scheduling a Report Overview

The Scheduled Reports screens lets you view all of the reports currently scheduled to run on the system, add a new report to the list, edit an existing report schedule, edit an existing report options, or enable/disable an existing report scheduled to run. You can configure the reporting interface to generate a report on a custom schedule which will be automatically emailed to any email address specified.

Figure 5: Scheduled Reports

Scheduled Reports								Add Scheduled Report	
Schedule Name	Report	Run	Period	Format	Options	To	Actions		
My Scheduled Scorecard	Scorecard	Weekly	(not applicable)		time_zone=UTC	executive@example.com	  		
My Country Report	Country Comparison Over Time	Monthly	1 Month		top=10&time_zone=UTC&countrycodes=AO,AR,AU	threats@example.com	  		

PART 2

Configuration

- [Initial Tasks on page 31](#)
- [Configuration Wizard on page 39](#)
- [Command Line Interface on page 47](#)
- [Configuration Options on page 51](#)
- [Clustering on page 61](#)
- [High Availability on page 65](#)
- [SRX Integration on page 71](#)
- [Reports on page 77](#)

CHAPTER 11

Initial Tasks

- [Initial Appliance Configuration on page 31](#)
- [TUI Steps on page 31](#)
- [Setting the Hostname on page 32](#)
- [Interface Configuration on page 33](#)
- [Set DNS on page 34](#)
- [Restart the network on page 35](#)
- [Initialize the Appliance on page 35](#)
- [Verify Connectivity on page 36](#)
- [Install the License on page 37](#)

Initial Appliance Configuration

Initial configuration is done through the console in a limited shell. Once WebApp Secure has been initialized for the first time, you can log into a Web console to finish the initial setup. Once the setup is complete, WebApp Secure is protecting your applications. You must use the direct console interface to configure the appliance IP address. Once the system has an IP address, you can use SSH to connect via port 2022.

Use the SSH command.

```
ssh <machine-ip-address> -p 2022 -l mykonos
```

- User: mykonos
- Password: mykonosadmin

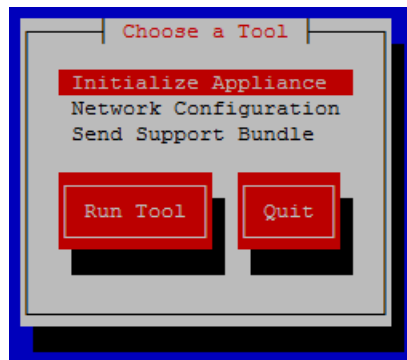


NOTE: You should immediately change these defaults after you login.

TUI Steps

The following steps are performed through the Text User Interface. This is initially done via the serial console, however the TUI is available through SSH once the Appliance is operating.

Figure 6: TUI, Initialize Appliance



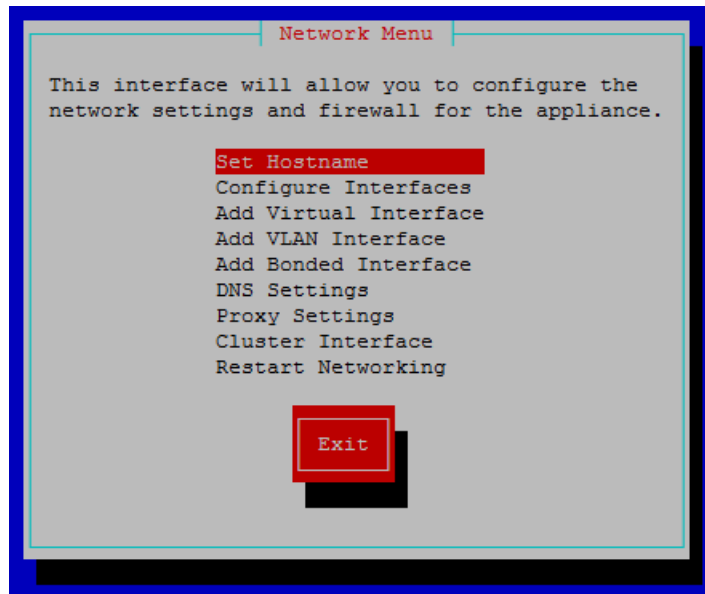
The minimum requirements for the appliance's network setup are to set a hostname and configure a network interface.

Setting the Hostname

To configure the network interface, do the following:

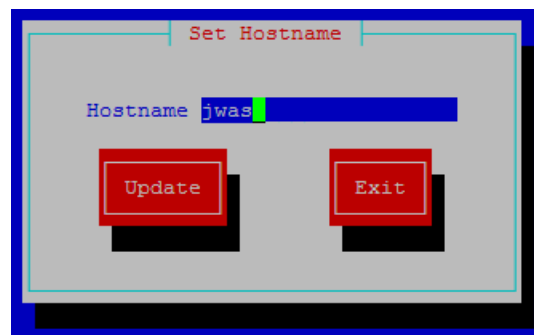
1. Enter the TUI and select **Network Configuration** from the menu.
2. Enter the **Set Hostname** menu.

Figure 7: TUI, Set Hostname



3. Enter the system's hostname and select **Update**.

Figure 8: TUI, Enter Hostname



Interface Configuration

Once the hostname is set, open the **Configure Interfaces** menu and enter the network settings for the appliance's network interface as follows:

1. Interface - Select network interface
2. IP Address - Specify a fixed IP address (if DHCP is unchecked)
3. Netmask - Specify the netmask
4. Gateway - Specify network gateway IP address
5. On Boot - Select to start the interface at boot time

Figure 9: TUI, Configure Interface

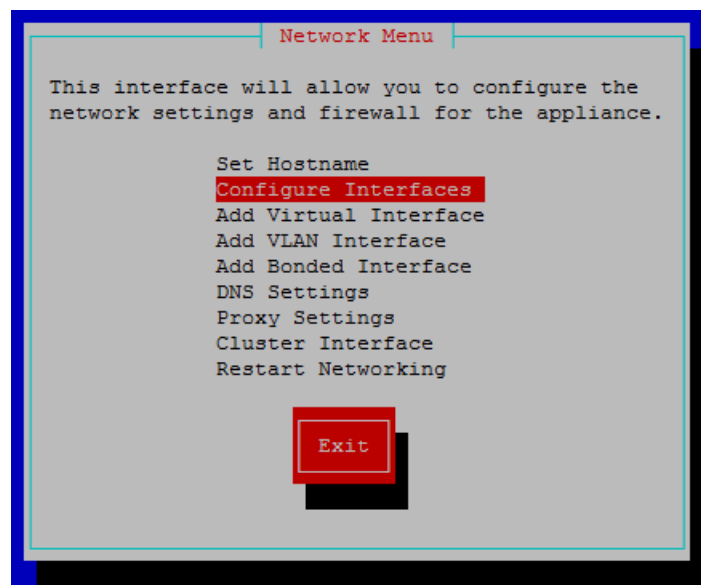


Figure 10: TUI, Network Interfaces

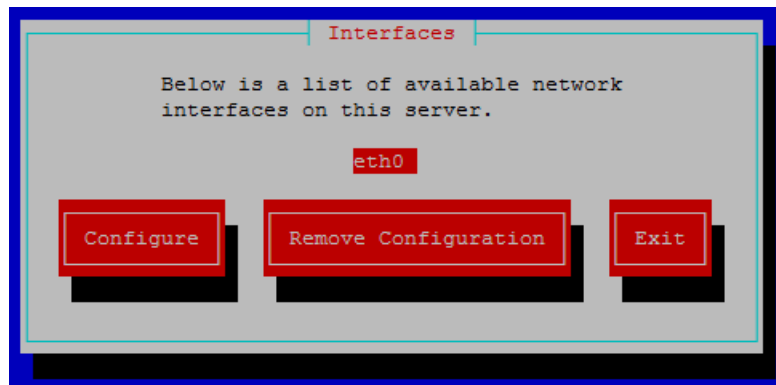
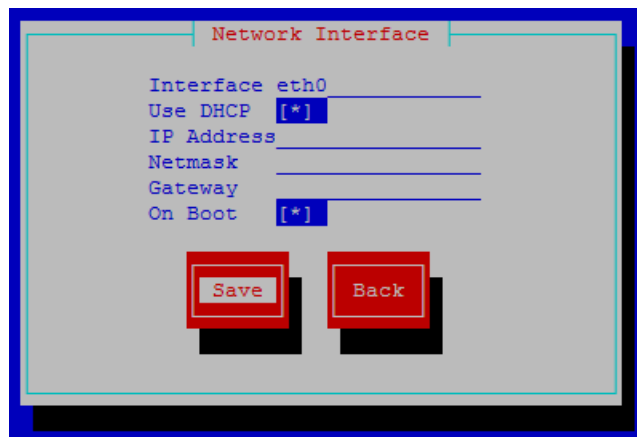


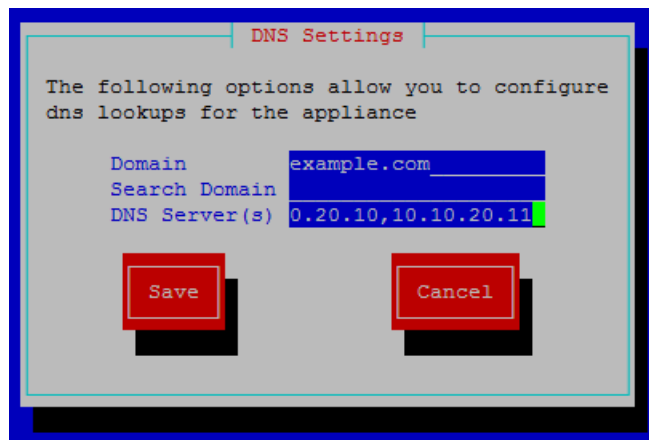
Figure 11: TUI, Configure Interface



Set DNS

Select DNS settings from the network menu, and enter appropriate values.

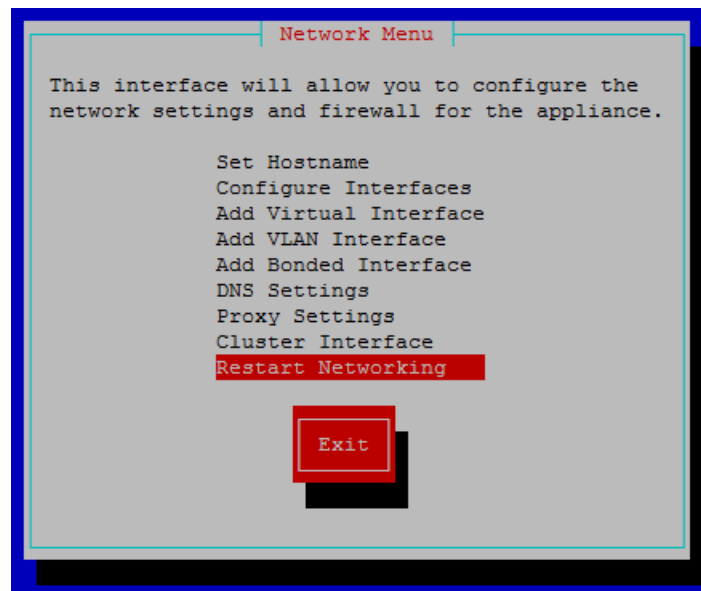
Figure 12: TUI, Set DNS



Restart the network

After the Hostname and network interface information has been set, you must restart networking to have the changes take effect.

Figure 13: TUI, Restart Network



Initialize the Appliance

To initialize the appliance's services, open the TUI menu and select the **Initialize Appliance** from the menu.

Figure 14: TUI, Initialize Appliance

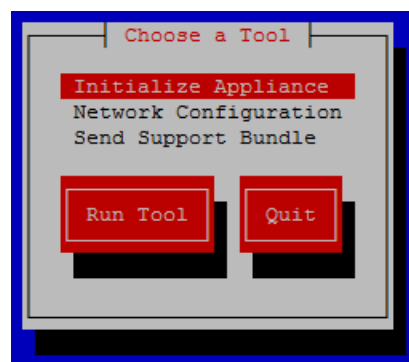
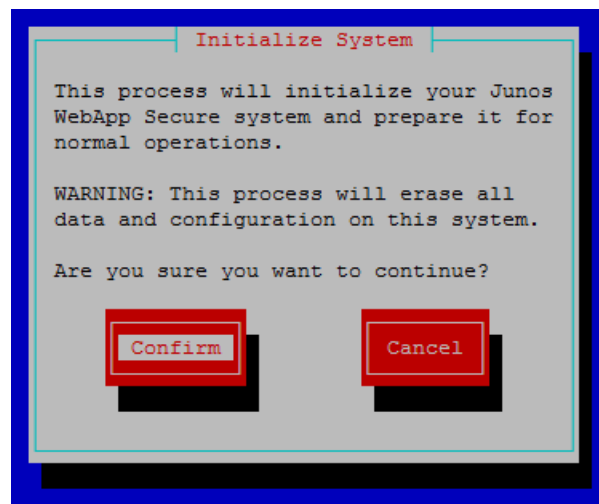


Figure 15: TUI, Initialize System



The Appliance typically takes 3 to 5 minutes to initialize, depending on available resources. Once initialization is complete, the administrator will have access to the Web interface where they can finish the initial configuration.



NOTE: Initialization sets the system to a default state and will overwrite any existing configurations. If this is the first time you're configuring the Appliance, it will not be an issue. However, if you are rerunning the initialization on an existing system, be aware this will reset it to defaults, deleting any existing data on the Appliance.

Verify Connectivity

Purpose After all initialization steps have been performed, verify that all network settings are correct, and that the appliance can be reached from the network. Navigate to the IP or hostname assigned to the appliance (on SSL), and specify port 5000.

Action For example:

- `https://10.10.10.104:5000`
- `https://my-hostname:5000`

If you see the appliance login screen, network settings are correctly configured.

Figure 16: Appliance Login Screen



Install the License

In order to complete WebApp Secure configuration, you must install the license for the product. Use a web browser to connect to your appliance on port 5000 and log in using the Administrator credentials.



NOTE: The configuration url is: `https://<IP address or hostname>:5000`

Examples: `https://10.11.12.13:5000` and
`https://webappsecure.mydomain.com:5000`

Go to the licensing section and follow the prompts:

1. Enter the license key provided to you in the **Add a New License** field.
2. Click **Add**.
3. Review the Terms of Service.
4. Click **Yes** on "I agree to the terms of service" to enable your WebApp Secure product.

Figure 17: License Terms

Licensing

License Key: MSAX-D3ED-17AD-5F3C-18AD

Please read the licensing terms before proceeding.

Juniper Networks License Terms

1. Definitions

In addition to terms defined elsewhere in this Agreement, this Agreement uses the following defined terms.

1.1 **"Approved Hardware"** means any physical device, system, sub-system, or component thereof that is certified by Juniper for the purposes of operating Licensed Software and/or Juniper Technology

1.2 **"Bug Fix"** means any modification or addition to any Licensed Software by Juniper, which is intended to correct Errors or other unwanted or unintended conditions that cause the Licensed Software to fail, malfunction, or operate in a manner other than as anticipated or desired.


1.3 **"Documentation"** means the technical manuals, user guides, and other information, if any, which are made available by Juniper to Licensee with respect to the Juniper Technology or other Licensed Software, together with modifications and updates thereto, whether in printed or machine-readable form.

1.4 **"Enhancement"** means any technology that enhances, improves, or otherwise modifies any Licensed Software.

1.5 **"Error"** means an error in any Licensed Software which significantly degrades the Licensed Software as

☒ I agree to the terms of service

☒ Send hardware details to Juniper customer support?

 **Save License**

If the license validation step fails, check the network settings, particularly proxy settings for the network. WebApp Secure must reach the outside world to contact the licensing server.

CHAPTER 12

Configuration Wizard

- [Configuration Wizard: Backend Server on page 39](#)
- [Backend Server Requirements on page 39](#)
- [SMTP Servers on page 40](#)
- [Wizard Alert Service on page 41](#)
- [Email Alert Requirements on page 42](#)
- [Wizard Backup Service on page 43](#)
- [Backup Service Fields on page 44](#)
- [Wizard: Spotlight Secure on page 44](#)
- [Wizard Confirmation Page on page 44](#)

Configuration Wizard: Backend Server

WebApp Secure functions as a reverse proxy, positioned in front of your web application. WebApp Secure can protect an arbitrary number of application servers. But in order to process traffic, you must specify at least one backend server to which the application will proxy traffic. The default is 1, and this should suffice in most situations. If WebApp Secure is serving as a software load balancer, rather than using a dedicated hardware solution, multiple servers can be configured at this time.

Figure 18: Wizard, Configure Backend Servers, Step 1



Backend Server Requirements

Each backend server you configure here requires the following information:

- **Server Name:** A unique name that WebApp Secure uses to identify this server. The name can include any alphanumeric character, "-", and "_", with no white space. Do not use the server's Fully Qualified Domain Name (FQDN) or a URL. If you are using VMware, you may wish to use the same name here as you assigned in VMware, to avoid confusion. But that is not required.
- **Server Address:** Specify the server's IP. WebApp Secure does not support IPv6 addressing at this time.
- **HTTP Port:** Usually port 80.
- **HTTPS Port:** Usually port 443.
- **Weight:** The default is 1. This value is used when WebApp Secure is serving as a software load balancer and represents the relative weight the server has for balancing purposes.
- **Backup:** The default is NO. This only applies if you are using WebApp Secure as a software load balancer, and you are designating this server as a backup.

Figure 19: Wizard, Configure Backend Servers, Step 2

STEP 2 OF 5: BACKEND SERVERS

For each backend server, specify a server name (just for your reference), and IP address and port information. If you'd like to use Mykonos Web Security as a simple software load balancer, you will also need to specify a weight other than 1, and whether or not each server is a backup server.

Server Name:	Backend1 <small>A simple logical and unique name of the server using only letters, numbers, and the dash character with no whitespace (do NOT use a url or a fully qualified domain name for this)</small>
Server Address:	10.10.10.101 <small>Specify a hostname or IPv4 address.</small>
HTTP Port:	80 <small>The TCP port to use for unencrypted (HTTP) communication with the server.</small>
HTTPS Port:	443 <small>The TCP port to use for encrypted (HTTPS) communication with the server.</small>
Weight:	1 <small>If using Mykonos Web Security as a software load balancer, this is the load balancing weight (higher = more requests).</small>
Backup?	<input type="checkbox"/> NO <small>If using Mykonos Web Security as a software load balancer, whether or not this server is a backup server.</small>

SMTP Servers

WebApp Secure can email alerts to your administration team. While the appliance can serve as its own mail server, it is recommended that you use a valid mail server for your network. SMTP configuration supports the following fields:

- **SMTP Default Sender:** This will be the "From:" address in email alerts. The address should be valid for your network so alert mails won't be incorrectly categorized as spam.
- **SMTP Server Address:** Defaults to localhost. Set it to the IP address or FQDN of your mail server if you are using an off-board mail server as recommended.
- **SMTP Server Port Number:** Defaults to 25. Set it to the port your mail server is listening on.

- **SMTP Username:** Defaults to blank, and may remain blank if you are using the on-board server. Set it to a user with valid access to the mail server.
- **SMTP Password:** Defaults to blank, and may remain blank if you are using the on-board server. Set it to the password for the SMTP username.

Figure 20: Wizard, Configure SMTP Settings, Step 3

STEP 3 OF 5: SMTP SETTINGS

Various components of Mykonos Web Security send email. If you'd like to use the built-in SMTP server, you can just click 'Next', but if you have an MTA on your network already, please specify the details below.

SMTP Default Sender:	<input type="text" value="support@mykonossoftware.com"/> <small>Will be used as the default 'from' address for email sent by the server. Address should be a valid sender for your domain to avoid being miscategorized as spam.</small>
SMTP Server Address:	<input type="text" value="localhost"/> <small>Enter the IP address or FQDN of your SMTP server.</small>
SMTP Server Port:	<input type="text" value="25"/> <small>If your SMTP server is on a non-standard port, enter it here.</small>
SMTP Username:	<input type="text"/> <small>If your SMTP Server requires authentication, enter the username here.</small>
SMTP Password:	<input type="password"/> <small>If your SMTP Server requires authentication, enter the password here.</small>

Wizard Alert Service

WebApp Secure can send alerts to an SNMP server or by email to appropriate personnel. The alert service is optional, and defaults to No. If you choose not to activate alerts, the Wizard skips to the next section.

Figure 21: Wizard, Configure Alert Service , Step 4

STEP 4 OF 6: ALERT SERVICE

Mykonos Web Security can be configured to send email alerts or SNMP traps to administrators when an incident of a specified severity is detected. For instance, we could send out an email notification to a specific admin who is on-call if an incident level of 'critical' is detected, allowing the admin to respond quickly to the threat.

Send Alerts?	<div> <input checked="" type="checkbox"/> YES </div> <small>The alert service can send SNMP traps or email system administrators when certain activity is detected. If you leave Alerts turned off, these will be logged *only*.</small>
---------------------	--

Figure 22: Wizard, Configure Alert Service , Step 5

STEP 5 OF 6: ALERT SERVICE

If you'd like to receive SNMP traps when alerts happen, enter the number of SNMP servers that will receive these traps, or '0'. If you'd like to have administrators receive email notifications when alerts happen, enter the number of administrators who will receive these alerts, or '0'.

Number of SNMP Servers:	0 <small>If you'd like to send SNMP traps, how many servers will receive them?</small>
Number of email contacts:	1 <small>If you'd like to send alerts via email, how many administrators will receive them?</small>

back next

If you choose to activate alerts, you have the option of setting up the number of SNMP servers to alert and the number of email addresses to which messages are sent. The default values to both are 0.

Figure 23: Wizard, Configure Alert Service , SNMP, Step 6

STEP 6 OF 8: ALERT SERVICE SNMP SETTINGS

For each SNMP server, please enter an IP address or FQDN, and a port, if other than the default.

Server Address:	 <small>Specify a hostname or IPv4 address.</small>
Port:	162

back next

If you activate SNMP Alerts, the wizard prompts you for the server address and the port to which alerts are sent.

Email Alert Requirements

Email alerts require the following fields:

- **Name:** A common name for referencing this email address.
- **Email Address:** Email address.
- **Minimum Severity:** Minimum severity level to trigger an email alert to this address.
- **Shift Start:** Start time for this address in 24 hour format.
- **Shift End:** End time for this address in 24 hour format.

You are also given the option of having alerts sent on the weekend. You can build complex schedules by creating multiple entries for the same person. For example, admin@yourcompany.com could have an entry named admin-weekday that specifies 8 AM to 5 PM, M-F, and a second entry named adminweekend that specified 6 AM to 6 PM.

Figure 24: Wizard, Configure Alert Service , Email Contacts, Step 7

STEP 7 OF 8: ALERT SERVICE EMAIL CONTACTS

Please specify the details for each Alert Contact, including their approximate on-call schedule. This information can be changed at any time, after the wizard is completed.

Name:	<input type="text"/>
Email Address:	<input type="text"/>
Minimum Severity:	<input type="button" value="High"/> <small>The minimum severity that the contact will receive alerts for.</small>
Shift Start:	<input type="text"/> <small>Earliest time that alerts will be sent to this contact (use 24-hour time).</small>
Shift End:	<input type="text"/> <small>Latest time that alerts will be sent to this contact (use 24-hour time).</small>
Sunday:	<input checked="" type="checkbox"/> YES <small>Whether or not alerts will be sent to this contact on Sundays.</small>



NOTE: Configuration of advanced features, such as encryption keys, are not available in the wizard.

Wizard Backup Service

WebApp Secure can perform regular, scheduled backups of all data. It is strongly recommend that you turn backups on. You can select backups using FTP or SSH.

Figure 25: Wizard, Configure Backup Service

STEP 7 OF 7: BACKUP SERVICE

Please specify how you would like backups to be conducted.

Frequency:	<input type="button" value="Weekly"/> <small>How often would you like backups to be performed?</small>
Retention:	<input type="text" value="30"/> <small>Number of days that backups will be kept before being deleted.</small>
FTP:	<input type="checkbox"/> NO <small>Would you like to push backups to an FTP Server?</small>
FTP Server:	<input type="text"/>
Username:	<input type="text"/>
Password:	<input type="text"/>
SSH:	<input type="checkbox"/> NO <small>Would you like to push backups to an SSH Server (via SCP)?</small>
SSH Server:	<input type="text"/>
Username:	<input type="text"/>
Password:	<input type="text"/>

Backup Service Fields

The backup service lets you specify the following fields:

- **Frequency:** How often backups are sent off-board.
- **Retention:** Number of days to keep off-board backups.
- **FTP Service:** Whether to use FTP. If set to YES, the server, username, and password fields are required.
- **SSH Service:** Whether to use SSH. If set to YES, the server, username, and password fields are required.

Wizard: Spotlight Secure

Spotlight Secure provides a way to import malicious profiles from other subscribers to the service. The service is enabled by default, but you can choose to disable it.

Figure 26: Wizard, Configure Spotlight Secure



STEP 6 OF 6: JUNOS SPOTLIGHT SECURE

Junos Spotlight Secure, licensed separately, is a new cloud-based hacker device intelligence service that will identify individual attacker devices and track them in a global database. If you are interested in leveraging this unique technology, please contact your account representative.

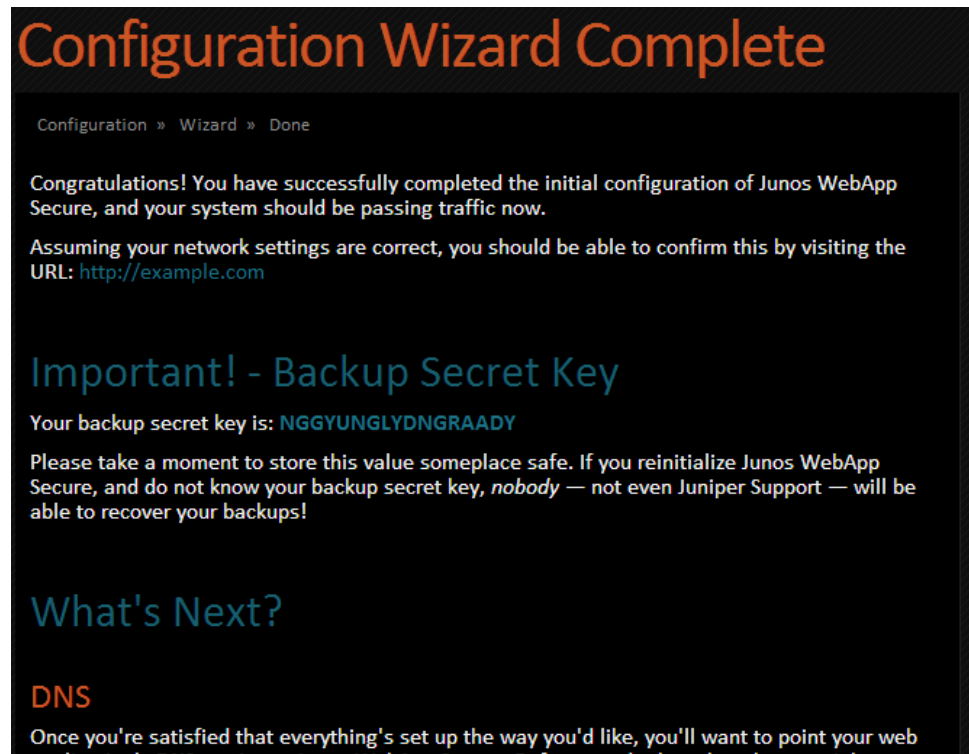
Enable Spotlight: ☒ Would you like to turn on Junos Spotlight Secure? (Licensed Separately)

←back next→

Wizard Confirmation Page

Once you have completed the wizard's main steps, you will see the confirmation page. Here, there is a URL you can use to confirm that the appliance is performing correctly. You will also see the secret key the appliance generated for your backups. Whether WebApp Secure is storing backups locally or off-site, you must have this key.

Figure 27: Wizard, Confirmation Page



NOTE: The key is actually a link. You must click on it and confirm your acceptance of the key.



NOTE: Record the secret key and keep it someplace safe. If you run through System Initialization again, it will create a new key and you will lose access to your backups if you haven't recorded the old key. If you lose this key, Juniper Support will not be able to recover it or your backups.



NOTE: It is also worthwhile to record other configuration entries in the event that you perform a configuration re-initialization. `engine.session.encryption_key` and `engine.session.initialization_vector` are entries needed to maintain the data of currently active users on the protected application. It is best practice to write these down, as well. Once configuration initialization is done, these old values may be set again.

CHAPTER 13

Command Line Interface

- Using the CLI on page 47
- CLI: Config on page 48
- CLI: Config: Setting a Configuration Parameter on page 48
- CLI: Config: Initializing the Configuration on page 49
- CLI: Config: Import/Export on page 49
- CLI: Config: Configure a Proxy Exclusion on page 50
- CLI: Services on page 50
- CLI: System on page 50

Using the CLI

While the Web-based configuration handles most configuration needs, the WebApp Secure CLI is also available for changing the appliance configuration. You can access the CLI via the shell, by entering **sudo mykonos-shell** from the command line.

There are currently three main systems within the CLI as follows:

- **config**: allows you to access the appliance configuration
- **services**: allows you to perform tasks on the various background services functioning on the appliance
- **system**: allows you to perform tasks on the various background services functioning on the appliance, and system allows you to shutdown or reboot the appliance



NOTE: At any point, you can enter **?** or **help** to view the available commands at the current context, or **help <command>** to get contextual help on the specified command. You can also show information on a particular parameter by using the **info** command. For example, **info services.cleanup.db.enabled**. Entering **Ctrl + D** or **exit** will leave the context you are currently in, or exit the CLI. The CLI also has full tab-complete support.

CLI: Config

Typing **config** at the CLI prompt will put the CLI into the configuration context. Configuration values are organized in a hierarchical fashion, with the most general words located at the beginning of the full configuration attribute string. For example:

```
services.cleanup.db.enabled
```

From the entry above, you can see that this parameter is for a service that handles the cleanup of the database. Specifically, this parameter determines whether the service is enabled or not.

Within the config context, you can choose to **show** any portion of the configuration. For example:

```
show services.cleanup.db.enabled
```

In the entry above, the value of the parameter in question is shown. If you want to see all of the configuration for the DB Cleanup Service, you can enter **show services.cleanup.db** to return a JSON object representation of the entire configuration that relates to the DB Cleanup Service. Likewise, entering only **show** displays the entire configuration as one large object.

CLI: Config: Setting a Configuration Parameter

To set a configuration parameter, enter **set PARAMETER.TO.SET VALUE TO SET TO**. For example, to enable the DataBase Cleanup Service (which allows you to delete profiles from WebApp Secure), enter the following:

```
set services.cleanup.db.enabled true
```

For more advanced users, you can edit configuration entries with an actual editor. If you do this, you can append **| edit** to the end of the set command where your value would be. The shell will put you into a text editor (VIM by default) where you can make changes to the configuration values. This is convenient when editing the JSON representation of a set of configuration entries, such as `services.cleanup.db`. Make any changes and, in VIMs Normal Mode, enter **wq** to write and quit the editor. For more information about VIM and how to use it, consult the VIM Documentation.



NOTE: You can choose to show a portion of the configuration without setting it by using the keyword **show** rather than **set**. For example: **show services.cleanup.db** This displays all configurations related to the DataBase Cleanup Service.

After making any changes, you can compare the new configuration with the last-saved version by typing entering **| compare**. A diff is printed to screen, with **-** indicating original settings, and **+** indicating modified settings. For example, changing the DB Cleanup Service from true to false will yield:

```
- - - Original Settings
```

```

+++ Modified Settings
@@ -2635,7 +2635,7 @@
services.backups.retention: 7
services.backups.secret: WIB25lklsbMM3wOR
services.backups.ssh.enabled: false
-services.cleanup.db.enabled: true
+services.cleanup.db.enabled: false
services.cleanup.db.expiration.history: 2592000
services.cleanup.db.expiration.malicious: 10368000
services.cleanup.db.expiration.session: 2592000

```

CLI: Config: Initializing the Configuration

At some point, it might be necessary to reset all configuration entries to default values. To do this, you enter **config init** at the root context, or **init** and **commit** at the config context. Once you do that, all entries are reset to their factory defaults.



WARNING: Because configuration initialization resets every parameter to a default, you might want to record some entries before doing this. Specifically, **engine.session.encryption_key** and **engine.session.initialization_vector**. Those two entries are needed to maintain the correct session data for currently-active users. If these values change, you might see false positives of Session Etag Spoofing incidents and Application Cookie Manipulation incidents, because the corresponding key values have indeed been manipulated.

CLI: Config: Import/Export

You can export a configuration image containing all configured parameters. This image can be imported by the system, letting you make a backup of your system configuration before making major changes, or to aid in some types of deployment. If you execute a system initialization from the TUI, you can use an imported configuration to bring your system back to its previous running configuration. However, historical traffic information is not part of the exported configuration and is therefore not recoverable.

To access the Configuration Import / Export feature, enter **sudo mykonos-shell** in an SSH session on the appliance, and at the prompt enter **config export <filename>**. The configuration is saved using the filename given. Similarly, import the configuration by entering **config import <filename>**.



NOTE: Because configurations can change from version to version of the product, importing a configuration exported from an older version of WebApp Secure may fail.

CLI: Config: Configure a Proxy Exclusion

WebApp Secure functions as a reverse proxy. Therefore, all web traffic goes through WebApp Secure so that it can analyze traffic for attacks. To improve performance, you can configure WebApp Secure to not process certain types of resources, such as images, or zip files, for example. To configure an exclusion for certain file extensions that you want routed around WebApp Secure (as opposed to through it), you can use the configuration setting described here.

For example, to have WebApp Secure not process any zip files, you can add the zip file extension to the proxy exclusions list by entering the following command at the WebApp Secure terminal:

```
sudo mykonos-shell set engine.proxy.exclusions zip
```

Now WebApp Secure will not process zip files.



NOTE: Google Web Toolkit utilizes specialized cache files that may conflict with WebApp Secure. If your protected site utilizes Google Web Toolkit, you will need to add the file extension of these cache files (typically .cache.html) to the engine.proxy.exclusions parameter.

CLI: Services

The services context lets you manage the various background processes that are running on the appliance. From this context, you can start, restart, and stop the services, as well as check the status of the services. An example output is given below:

```
service> status
nginx_management (pid 13749) is running...
mykonos-datastore (pid 13759) is running...
mykonos-api (pid 13844) is running...
mykonos-ui (pid 13866) is running...
mykonos-services (pid 13876) is running...
mykonos-cluster-services (pid 13924) is running...
mykonos-reports-api (pid 13960) is running...
mykonos-security-engine (pid 14347) is running...
```

CLI: System

Within the system context, commands exist to restore from a supplied backup, or reboot and shutdown the system.

Configuration Options

- [Securing Multiple Web Servers on page 51](#)
- [Create a New Application on page 51](#)
- [Edit Applications on page 52](#)
- [Application Patterns on page 53](#)
- [Define Backend Servers on page 55](#)
- [Enable SSL to the Client on page 56](#)
- [Whitelist Settings on page 57](#)
- [Configure Support for Akamai Dynamic Site Accelerator on page 58](#)

Securing Multiple Web Servers

In more complex environments, it might be required to separate functionality into multiple applications, each with their own settings and configuration. The **Applications** section of the UI can help manage multiple applications/URL patterns from a single WebApp Secure instance.

Create a New Application

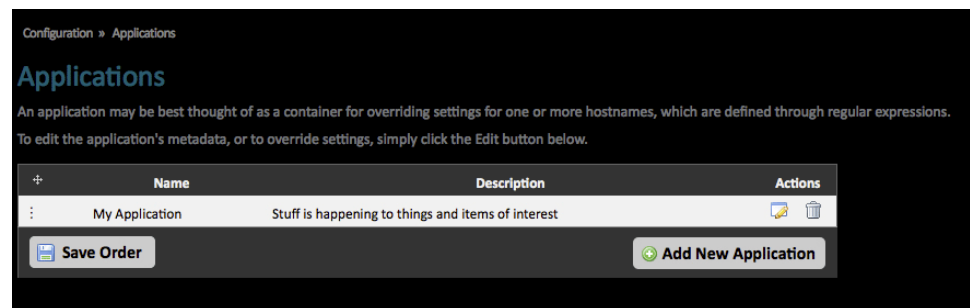
To create a new application within WepApp Secure, do the following:

1. In the Web UI, click **Configuration > Applications**. Here you can view applications that have already been configured.



NOTE: You can reorder applications in the applications list by dragging and dropping them. Click **Save Order** to preserve your newly ordered list.

Figure 28: Configured Applications



2. Enter information into the form to create a new application.
3. Click the **Add Application** button. This launches the Application Wizard. Fill in the available fields, and continue to click the **Next** button until you have completed the wizard. Based on your entries, the wizard can require three to six steps to complete.

Figure 29: Application Wizard

STEP 1 OF 3: APPLICATION METADATA

Please specify basic application metadata, including a friendly name, description, and a slug (short name used in the Configuration system).

Name:	<input type="text"/>	Please enter a human-readable "friendly name", to be used throughout the interface.
Slug / Short Name:	<input type="text"/>	Please enter a unique, URL-friendly, short identifier comprised of lower-case letters, numbers, and / or underscores ONLY.
Description:	<input type="text"/>	Please (optionally) enter a brief, human-readable description.
Number of Patterns:	<input type="text" value="1"/>	Most applications only need one pattern. If you are unsure, just leave this '1' here. You can always add more later.
HTTP Port:	<input type="text" value="80"/>	The TCP port this application should listen on for HTTP traffic (suggested value: 80).
Configure SSL?	<input type="checkbox"/>	If you would like your application to be available over SSL, check this box.

next→

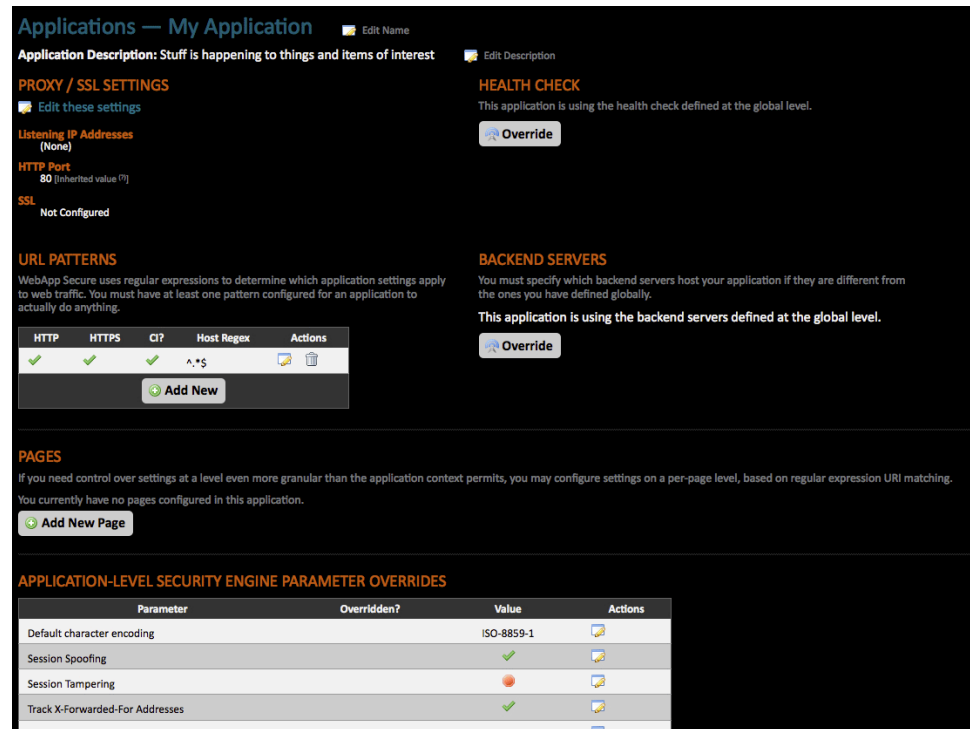
Edit Applications

You can edit an existing application by doing the following. If you want to change any configured application settings, click the **Edit/Override Settings** icon for the application in question.

1. Navigate to **Configuration > Applications**. Existing applications are listed in the Application screen.
2. To edit an application, click the **Edit/Override Settings** icon under **Actions** for the application in question.

- This takes you to the dashboard for the application. The dashboard displays which application settings are overridden and which are inherited. From here, you modify each setting individually.

Figure 30: Application Dashboard



Application Patterns

Application patterns determine which requests get routed to which applications. You can change each application you've added later by navigating to **Configuration > Applications**. Url patterns follow standard Perl Regular Expression (PCRE) syntax.

For example:

- Any traffic: `^.*$`
- Any subdomain: `^.*\.domain\.com$`
- Multiple, or no, subdomains: `^((www|shop)\.)*domain\.com$`

Figure 31: URL Pattern

URL Pattern

The patterns used to check the URL to see if it belongs to this application

SUGGESTIONS

Catch All

HTTPS Enabled
Whether to apply this pattern to HTTPS traffic

HTTP Enabled
Whether to apply this pattern to HTTP traffic

Host Regex

Host Case Insensitive



NOTE: WebApp Secure processes applications in order. Therefore conflicting regular expressions will only be processed on the application or page where it first appears. There are also some suggestions that cover some common use-cases such as catch-all, subdomains, etc.

Figure 32: Application Patterns

APPLICATION PATTERNS

Junos WebApp Secure uses regular expressions to determine which application settings apply to web traffic. You must have at least one pattern configured for an application to actually do anything.

https	http	host.regex	host.case_insensitive	Actions
true	true	^111[.]222[.]111[.]222\$	true	



WARNING: URL patterns and profiles are observed in the order they are created.



NOTE: If SSL is required for this application, you will also need to enable SSL and ensure that all the required certificates are uploaded and configured properly.

Define Backend Servers

When separating applications, one application may reside on a different physical server. You can define a backend server for this application here. For convenience, WebApp Secure imported the existing backend server that was used in the global context.



NOTE: You do not have to define a backend server at the global level. For example, you can define backend servers using only Applications. However, if there is a backend server defined globally, you should not unset it. You can only change it. There should always be at least one global backend server. Deleting the last global backend server may cause instability. (If you want to define backend servers using only Applications, you can skip this wizard setup step.)

Figure 33: Define Backend Servers

BACKEND SERVERS

You may specify the backend servers for this application here, or in the context of the "Security Engine" configuration screen. For convenience, we have inherited the value from the "Global" context. These may be deleted if desired.

name	address	ports.http	ports.https	weight	backup	Actions
ALogicalNameHere	10.10.20.99	80	443	1	false	

Add



NOTE: WebApp Secure supports different configurations for different pages within a protected application. Fill out the required information and click Add Page to create a new page.



NOTE: The page nomenclature is used for simplicity. Much like applications, page contexts can define a set of pages using a RegEx. They aren't restricted to one actual page on the application.

Figure 34: Add New Page

ADD NEW PAGE

Page Metadata

Name *
Please enter a human-readable "friendly name", to be used throughout the interface.

Slug / Short Name *
Please enter a unique, URL-friendly, short identifier comprised of letters, numbers, underscores, and / or dashes.

Description *
Please enter a brief, human-readable description.

URL Pattern

To add a page, you must define a URL pattern to match traffic against.

Suggestions:

HTTP? ☒
Whether to apply this pattern to HTTP traffic

HTTPS? ☒
Whether to apply this pattern to HTTPS traffic

URI Regex *
The regular expression to match on the URI

URI Case Insensitive? ☒
Whether the URI regex should be case-insensitive or not

Enable SSL to the Client

To enable SSL between WebApp Secure and the client, do the following:

1. In the Web UI, navigate to the application for which you want to enable SSL or switch to the desired application's context.
2. Navigate to **Configuration > Applications > My App > Proxy/SSL Settings** and enable SSL to the backend.
3. Upload your SSL certificate and key file.
4. Select a listening interface IP address and HTTP and HTTPS ports,



NOTE: The combination of port/IP must be unique for the system. If the system is clustered, an IP must be selected for each node.

5. When you save the SSL configuration in a deployment containing multiple appliances, the certificate is propagated from the master system to all subsequent systems.

Figure 35: Proxy / Backends

Proxy / Backends

HTTP Port
The TCP port this application should listen on for HTTP traffic (suggested value: 80).

HTTPS Port
The TCP port this application should listen on for HTTPS traffic (suggested value: 443).

IP Addresses
The IP address to listen on for HTTP and HTTPS. This list is populated automatically based on the IP addresses you currently have assigned to interfaces.

SSL Certificate
Since the system needs to decrypt SSL traffic to work, you will need to copy-paste your SSL Certificate (in PEM format) into this box.

SSL Key
Since the system needs to decrypt SSL traffic to work, you will need to copy-paste your SSL Private Key (in PEM format) into this box.



WARNING: To safeguard against inheriting SSL certificates, WebApp Secure does not allow SSL at the global level. Therefore, you must configure an application in order to enable SSL.



WARNING: Your certificate and key files cannot be password protected. If they are, WebApp Secure will be unable to read them. You can remove passwords on your existing certificate by using the openssl program. For example, `openssl rsa -in mykey.pem -out newkey.pem`.



NOTE: Certificates must be in valid PEM (Privacy Enhanced Mail) format. You can verify the SSL certificate by using the command, `openssl verify <sslcert.crt>`. WebApp Secure is only concerned with the validity of the format. openssl verify might allude to other problems with the certificate, but other issues do not come into play when used within WebApp Secure.

Whitelist Settings

There are various types of whitelists to which you can add IP addresses. To access the Whitelist screen, do the following:

Navigate to **Configuration > Security Engine > Whitelists**. The following types of whitelists are available:

- **Trusted IP Addresses:** The IP addresses in this list will not trigger incidents. Click **Add New** to enter IP addresses to be added to this list.
- **X-Forwarded-For Address Exclusions:** The IP addresses in this list are stripped off the X-Forwarded-For header. This effectively trusts that the next IP address in the chain is a genuine address. Click **Add New** to enter IP addresses to be added to this list.

Figure 36: Whitelists



Configure Support for Akamai Dynamic Site Accelerator

You can configure WebApp Secure to work with a site that utilizes Akamai Dynamic Site Accelerator. You will need to make minor changes to your site's configuration in the Akamai Luna Control Center and in the Content Delivery Network section of the Security Engine configuration screen in the Configuration UI.

To make the necessary changes, do the following:

1. Log into Luna Control Center and select the **Configure** tab.
2. Click the link corresponding to the desired site configuration under **Configuration Name**.
3. On the next screen, find the currently-active configuration and click **Create Version from...** in the right-hand column. Make the following changes:

Table 3: Luna Control Center Configuration Changes

Configuration Section	Parameter	Value
Honor HTTP Cache-Control and Expires Headers	Cache Control Headers	false (uncheck)
Honor HTTP Cache-Control and Expires Headers	HTTP Expires Headers	false (uncheck)

Table 3: Luna Control Center Configuration Changes (*continued*)

Configuration Section	Parameter	Value
Browser Cache Control Headers	Pass through the origin's Cache-Control headers to the browser	true (select)
Browser Cache Control Headers	Pass through all origin cache control headers	true (select)
Edge Services - General	Enable True Client IP Header	true (check)
Edge Services - General	True Client IP Header Name	True-Client-IP (or other; see below)
Edge Services - General	Enable Edge Server Identification	false (uncheck)



NOTE: Choosing a name for the True-Client-IP header other than the default may provide additional security by preventing malicious users from spoofing this header. Make a note of the value chosen for the header. You will need to configure it on the WebApp Secure side.

- After making these changes, scroll to the bottom of the page and activate the new Akamai configuration as you normally would.
- Once you have verified that your new Akamai configuration has gone live, log into the WebApp Secure web UI. If you are configuring Akamai support for an application, browse to that application's configuration page. Otherwise, browse to the **Content Delivery Network** section of the **Security Engine** configuration (or use the Configuration CLI). Make the following changes:

Table 4: WebApp Secure Configuration Settings for Akamai Support

Parameter ID	Parameter Name	Value
engine.cdn.akamai.enabled	Akamai: Enabled	true
engine.cdn.akamai.true_client_ip	Akamai: True-Client-IP Header	(value specified in Akamai configuration)
engine.cdn.akamai.incidents.spoofing.enabled	Akamai: Spoofing Incident Enabled	true or false

- Set **Akamai Enabled** to **true** and **True-Client-IP Header** to the value that you configured in the Luna Control Center.



NOTE: If you want a security incident to be triggered when a client attempts to spoof a request through Akamai, you may enable the Akamai **Spoof Attempt** incident. This incident carries a severity of Medium and may be incorporated into custom Autoresponse rules.

CHAPTER 15

Clustering

- [Using a Secure Cluster on page 61](#)
- [Setting Up Clustering on page 61](#)
- [Updating the Cluster on page 63](#)

Using a Secure Cluster

Individual WebApp Secure appliances have the ability to work together as one system in a cluster. Clustering allows traffic to be divided among multiple appliances, effectively reducing the per-system load. In a clustered network configuration, the master node holds the database that is populated by one or more traffic processors. In order to successfully utilize a WebApp Secure cluster, a load-balancer must properly segregate traffic to each of the defined traffic processing nodes. Each of these traffic nodes must maintain connectivity with the master in order to operate.



NOTE: Clustering should not be confused with High Availability. Clustering is used to increase throughput (by utilizing multiple processing nodes), and can reduce the chance that the whole system will fail. Clustering does not protect the master node from failure as in a High Availability setup; only HA configurations are set up to include failsafe procedures to designate a new master when the first one is unavailable.

Setting Up Clustering



WARNING: Unlike a traditional cluster, a WebApp Secure cluster does not automatically balance traffic between each of the nodes. For this reason, a load balancer is required to be configured to send traffic to each traffic processing node.

Setting up a cluster is as easy as configuring multiple stand-alone boxes. The first step is to set up the master. You must set up the master first because you will need to supply the master's IP when initializing the traffic nodes.

To initialize a master, choose **Master** or **Dedicated Master** from the TUI setup menu (**sudo setup**).

Figure 37: TUI, Initialize Appliance

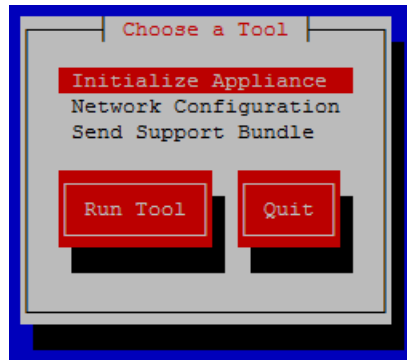


Figure 38: TUI, Select System Mode

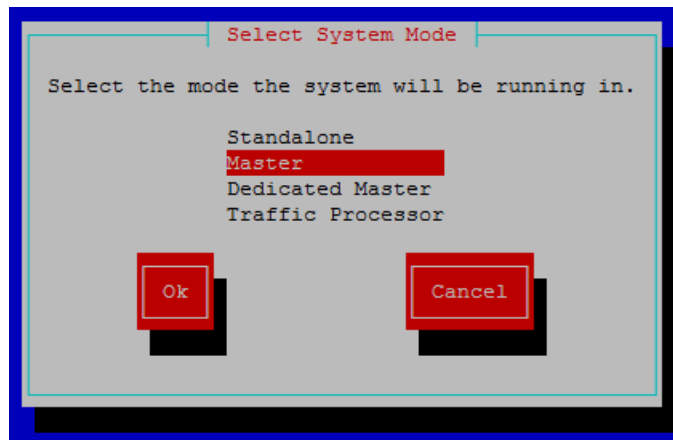
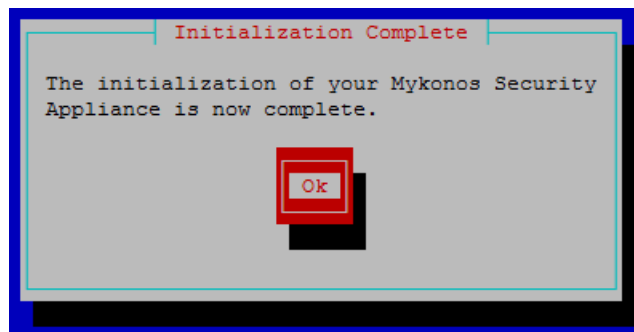


Figure 39: TUI, Initialization Complete



Setup will initialize the master, and once the initialization is complete, you should be able to navigate to the management interface at <https://HOSTNAME:5000>. Once the master is initialized, you can initialize the other appliances as traffic processing nodes. The steps are similar to the master setup, however you will be prompted to enter the IP of the master node.

Figure 40: TUI Select Traffic Processor

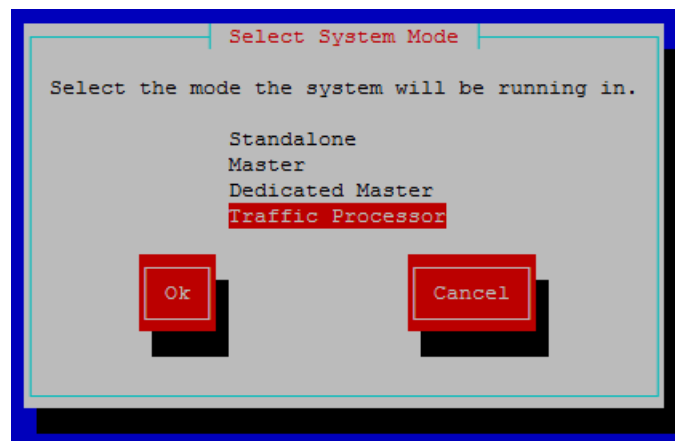
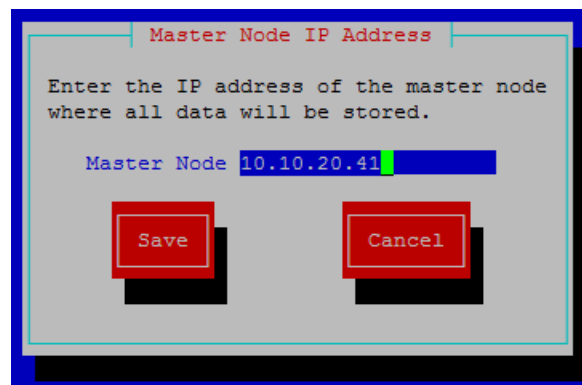


Figure 41: TUI, Master Node IP Address



Once the traffic node is initialized, you can verify the cluster by navigating to the management interface (<https://HOSTNAME:5000>) and clicking on **System Stats**. There should be a separate tab for each node in the cluster, and an additional tab for the aggregate cluster data.



WARNING: Remember that you must use an external load balancing solution to point to each traffic processing node, as the cluster will not do this for you.

Updating the Cluster

Updating a cluster is similar to updating a stand-alone box. Navigate to **Updates** in the management interface on the master node (the traffic nodes have no management interface) and apply the updates as you would on an individual appliance. The master node will automatically apply the same update to each of its traffic processing nodes in the cluster. There is no need to individually update each appliance.



NOTE: The process for updating a cluster will take longer than updating a single appliance since the same update must be applied to each node.

CHAPTER 16

High Availability

- [High Availability Overview on page 65](#)
- [Configuring High Availability on page 65](#)
- [Updating with High Availability on page 69](#)

High Availability Overview

To minimize the risk of downtime, WebApp Secure deployments have the ability to be placed in a Highly Available (HA) configuration. In this setup, an additional appliance is on stand-by in the event that the currently-active appliance goes offline. If this happens, the passive appliance is able to become the new active appliance automatically - without needing to restart the system. An HA configuration is similar to clustering, with the major exception being that the passive system has a copy of the services needed to take over when the master fails. WebApp Secure uses a Virtual IP (VIP) to float between the currently active system and the current passive system.



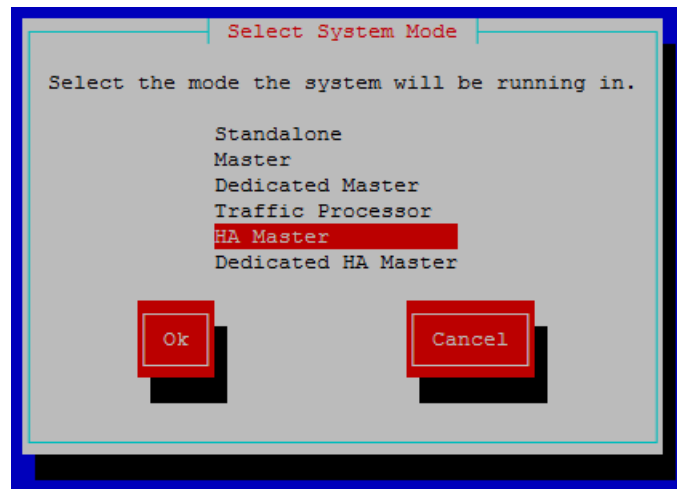
NOTE: An HA configuration is only available on WebApp Secure dedicated hardware systems. It is not available in a Virtual Machine installation.

Configuring High Availability

If your appliance is HA-ready, you will see two additional modes available on the **Select Appliance Mode** screen during appliance initialization.

1. On the active appliance (the one that will be the primary appliance), enter TUI setup by typing **sudo setup**.
2. Select **Initialize Appliance** and select either **HA Master** or **HA Dedicated Master** for the mode.

Figure 42: TUI, Select HA



The available modes are:

- **HA Master:** If the appliance is in HA Master mode, it acts as a stand-alone system. The system's database will be stored on this node, and will be replicated/mirrored to the passive appliance (configured later). The appliance will also process traffic like a standard installation.
 - **HA Dedicated Master:** Just like in clustering, the Dedicated Master has no way to process traffic by itself. It only contains the database and the essential services to talk to the other appliances. If you want a WebApp Secure cluster configured for HA, you can select this mode to prevent the master from processing any traffic. Keep in mind you just utilize clustering to configure at least one Traffic Processing node. A copy of the master will still exist on the passive system.
3. Once a mode is selected, you must bind the appliance to an interface. This must be the same interface that will accept incoming connections to the appliances and the same interface that the VIP is on. The HA interconnect interface can only be used as a link between HA appliances.
 4. After selecting the interface, you are prompted to enter the IPs belonging to the HA pair. This includes the IP of the current master (the active appliance) as well as the IP of the appliance to fail-over to (the passive appliance).



NOTE: Be sure that each of the appliances are on the same WebApp Secure version (invoke `mykonos-get-version` from the appliance's command line). Appliances not on the same version as the master must be manually updated to the HA master's version before continuing on.

Figure 43: TUI, Enter HA Node Addresses

- Next, you are prompted to enter the Virtual IP (VIP) that the system will use as the IP of the currently active system. You may enter either the standard or the CIDR bitmask (for example, 255.255.255.0 or /24) for the netmask.

Figure 44: TUI, Enter Virtual Addresses

- After allowing the Initialization process to complete, you can verify proper HA setup by navigating to the management interface <https://VIP:5000> where VIP is the Virtual IP. Navigate to **High Availability** on the left-side menu to observe the status of the HA pair.

Figure 45: HA Pair Status


Node Information

	ha-1	ha-2 (localhost)
Online?	Yes	Yes
Role	Secondary	Primary
Device State	Connected	Connected
Disk State	UpToDate	UpToDate

Performance Indicators

Network Send (NS) ^(?)	105 bytes
Network Receive (NR) ^(?)	151.0 KB
Disk Write (DW) ^(?)	151.1 KB
Disk Read (DR) ^(?)	154 bytes
Activity Log (AL) ^(?)	57
Bitmap (BM) ^(?)	9640
Local Count (LO) ^(?)	0
Pending (PE) ^(?)	0
Unacknowledged (UA) ^(?)	0
Application Pending (AP) ^(?)	0
Out of Sync (OOS) ^(?)	0 bytes

Manual Failover

 **Initiate Manual Failover**



WARNING: Since the various HA appliances in a configuration need to interface with the database, port 5432 will be open. Be sure to restrict access to this port with your firewall to prevent unwanted incoming connections. WebApp Secure is not intended to be used as an edge device.



NOTE: If the interconnect between an HA pair drops at any point, it is possible that both systems will try to assume the active system role. This leads to a condition known as split-brain, where data is not properly routed through the pair. To mitigate this, it is recommended that you bond the pair using the 10Gb ports on the front of the appliance. Follow the steps in the Network Configuration section of the TUI to setup the bond and then to configure it as you would any other interface.



NOTE: You must use the VIP to access the configuration interface. If you attempt to use the management interface on the passive appliance, you will see a notification indicating "The Administrative interface is not accessible on this host because it is the secondary host in a High Availability cluster."

Updating with High Availability

To update an HA system, navigate to the management interface (<https://VIP:5000> where VIP is the Virtual IP) and update as described in the System Updates section. The update will be applied to both systems in the HA pair.



NOTE: While both the active and passive machines must be on the same WebApp Secure version to be initially configured in HA mode, appliances already in HA mode will successfully update together.

SRX Integration

- [Filters and Terms Configuration Summary for SRX Integration on page 71](#)
- [Creating SRX Filters and Terms on page 72](#)
- [Configure the SRX Integration on page 73](#)

Filters and Terms Configuration Summary for SRX Integration

The SRX uses a pipeline of filters to be applied to incoming packets. Each filter contains any number of terms that can apply actions to these incoming packets. The first step in configuring WebApp Secure to work with the SRX is to configure the filters and terms required. WebApp Secure will require a valid IPv4 filter. This can be named anything and can be a filter already set up prior to WebApp Secure integration. Remember this filter name, because you will input it into the WebApp Secure appliance once the SRX configuration has been completed.

Along with a filter, you must create two terms. Unlike the filter, these terms cannot be modified by any other service. The first term is the term that IP addresses are added to in the event of an External Counter Response activation, and whose name will be supplied to configuration. The second term must be added as a safeguard which will determine what action to take when no IPs are in the first term. It is recommended that the second term be similar to the following:

```
term jwas_default {  
  then {  
    accept;  
  }  
}
```

This should be placed after the **blocking term**. It allows all traffic through once the previous term's action has been changed to **next term**. Consult the SRX documentation for more information on the SRX and its filters.



NOTE: Because the SRX will drop packets when **next term** is the action and no actual next term exists, it is important to have this additional term below the term which will contain the actual IPs.



WARNING: Any IPs added to the WebApp Secure term through the SRX CLI, the SRX GUI, or any other external service besides WebApp Secure, are not guaranteed to remain in the term.

Creating SRX Filters and Terms

To initialize a filter for use with WebApp Secure do the following:

1. Log into the SRX via SSH. Then enter **cli** and next enter **configure** to put the cli into configuration mode.

Figure 46: Initialize Filter

```

10.10.20.59 - PuTTY
root@% cli
root> configure
Entering configuration mode

[edit]
root#
  
```

2. Next create the filter, term, and a placeholder action. Because each term must have some sort of action, choose the **next term** action. This passes the packet on to the next term in the filter. Although the inside of the term will be replaced by WebApp Secure, it will allow the filter to be created. To do this enter **set firewall family inet filter my_filter term block then next term**. You can enter **show firewall** to see your newly-created filter.

Figure 47: Create Filter Term

```

root@my-srx
root# ...net filter my_filter term block then next term

root# ...net filter my_filter term default then accept

[edit]
root# show firewall
family inet {
    filter my_filter {
        term block {
            then next term;
        }
        term default {
            then accept;
        }
    }
}

[edit]
root#
  
```



NOTE: The filter name `my_filter` and term `block` are example names. You may choose any names you like, but remember them because you will need to inform WebApp Secure of your name choices later on in the configuration.

- Although the filter is created, it is not set to intercept incoming packets. You must now bind the filter to an interface. The interface and unit names will be different depending on your network implementation, but an example is: **set interfaces ge-0/0/0 unit 0 family inet filter input my_filter**. After binding to an interface, you should see the newly created filter appear under the appropriate interface when you enter **show interfaces**.

Figure 48: Bind Filter to Interface

```

root@my-srx
root# set interfaces ge-0/0/0 unit 0 family inet filter my_filter

[edit]
root# show interfaces
ge-0/0/0 {
  unit 0 {
    family inet {
      filter {
        input my_filter;
      }
      address 10.10.20.59/24;
    }
  }
}

[edit]
root#
  
```

- Save the changes by entering **commit**. Exit the CLI by entering **exit** twice (once to exit configure mode, and once to exit the CLI).



WARNING: If the blocking term is misplaced after the default (accept) term, the filter will not commit. Make sure that the accepting term is placed after the blocking term. Remember: next term needs a next term to switch to.

Configure the SRX Integration

To configure the integration of an SRX appliance with WebApp Secure, you must enable the External Counter Response Service, found within the configuration of the WebApp Secure web interface. The External Counter Response Service allows the SRX to send filter requests to the Appliance, and can be found under the **Global** section of the WebApp Secure configuration. It is an **Advanced** configuration set, so you will need to show the advanced configuration entries to see the External Counter Response Service configuration category.



WARNING: The configuration category will validate if there is an IP address or hostname in the corresponding configuration entry, and a filter name along with a term name, but this does not mean the service is properly working. Always test the counter response after changing the configuration entries, explained in the next section.

Be sure to examine the configuration entries available for this service, and fill out all necessary fields, outlined in the following table.

Table 5: External Counter Response Service Configuration Parameters

Parameter	Type	Default Value	Description
External Counter Responses Enabled	Boolean	False	Whether or not to enable this service.
Network Address	IP (or DNS name)	[Not Set]	Required. Either the IP address or the DNS name of the device.
SRX Password	String	[Not Set]	The password to log into the SRX.
SRX Username	String	[Not Set]	The username to log into the SRX.
Filter Name	String	[Not Set]	Provide a filter name that WebApp Secure will use.
Term Name	String	[Not Set]	The term in the configured filter that WebApp Secure should add the IPs to. It should not be currently in-use by any other service, and should only be used for WebApp Secure.
Action(s) to Apply)	Collection (Strings)	[collection:1]	Choose the actions for the SRX to take on IPs sent to it by WebApp Secure. When no IPs are blocked on the SRX through WebApp Secure, these terms will be changed to Evaluate Next Term , which will continue to the next term in the filter. By default, this is set to a collection of 1, consisting of only discard .



WARNING: When configuring multiple actions to take, be careful not to populate the collection with conflicting actions. An example of two conflicting actions are **reject** and **accept** (You cannot reject a connection and then accept a connection!). WebApp Secure has no protection for conflicting actions. The system will overwrite older actions with newer ones (further down the collection). An example of non-conflicting actions are **log** and **discard**. In this case, the packets will be logged, and then discarded. For more information on actions to take, consult the SRX documentation.



.....

NOTE: If the External Counter Response Service is disabled or otherwise configured incorrectly, blocking a profile via the External Block response will not work, but will still be shown in the User Interface as a valid Counter Response.

.....

CHAPTER 18

Reports

- [Schedule a Report on page 77](#)
- [The Reports CLI on page 79](#)
- [Supported arguments on page 80](#)
- [Data Sources on page 81](#)
- [Formatting on page 81](#)
- [Example Report on page 82](#)

Schedule a Report

To schedule a report, do the following:

1. Select the **Schedule Report** left navigation link.
2. Click the **Add Scheduled Report** button at the top right of the Scheduled Reports page. This brings up a list of reports to run. Choose the report that you want to run on a repeated basis by following its link.
3. On the subsequent page, enter all of the schedule details and report options and then select **Generate Report Schedule**.
4. **Save** the changes.

Most reports share the following items:

- **File type:** The file format that will be used to generate the report. Options usually include PDF or CSV. Certain reports are only available in PDF.
- **Schedule Name:** The name of the report schedule that will appear in the reporting interface.
- **Run:** The time schedule in hours, weeks, months, or years that the report should run on.
- **Period:** The period of time that the report should be run on.
- **Send to:** The email address that this report should be sent to.
- **Enabled:** Sets this report schedule to active (YES) or inactive (NO). Inactive reports will not be run on a scheduled basis.

Figure 49: Schedule Report - Scorecard

Reports » Schedule Report » Scorecard

Schedule Report: Scorecard

File type pdf ▾


Timezone * UTC ▾
Timezone to be used as selection range and display for the dates/times in the report. You may adjust your default timezone in [User Preferences](#). Please note that if you are planning on having this report emailed to people in different timezones, it may make more sense to create two separate scheduled reports.

Schedule name * My Scheduled Scorecard
Pick a name for the scheduled report.

Run * Weekly ▾
How often would you like this report to run?

Send to executive@example.com
Separate email addresses with a comma.

Enabled ☒

 **Add Report Schedule**



NOTE: Individual reports may have various additional options that are specific to that report. For instance the Country Comparison Over Time report contains a field for the number of countries to show and a list of specific countries to include.

Figure 50: Schedule Report - Country Comparison Over Time

Schedule Report: Country Comparison Over Time

File type: pdf

The number of countries to show (top n): 10

Countries to include: Argentina, Australia, Belgium

Timezone *

Schedule name *

Run *

Period *: 1 Hour

Send to:

Enabled: ☐

Country Selection List:

- ☒ Check all
- ☒ Argentina
- ☒ Australia
- ☐ Austria
- ☒ Belgium
- ☐ Bosnia and Herzegovina

Add Report Schedule

The Reports CLI

WebApp Secure contains a Command Line Interface (CLI), which provides users with access to the primary data sources. The CLI allows users to generate complex reports based on the data gathered by the security engine for the purpose of reporting and data analysis. The interface can be run by executing the following command:

```
sudo mykonos-reports-cli
```



NOTE: To get a list of required and optional arguments, refer to the man pages at: `man mykonosreports- cli`

Figure 51: Reports CLI

```

Mykonos Command Line Interface:
The mykonos command line interface provides access to the raw data collected by
the security engine. For a full description of what this command can do and how
it works, reference the man page.

    man mykonos-reports-cli

Syntax:
mykonos-reports-cli
mykonos-reports-cli -l
mykonos-reports-cli -d=DATASOURCE -h
mykonos-reports-cli -d=DATASOURCE [OPTIONS]

Arguments:

-l
    get the list of all supported data sources and their names

-d=<data source>
    specify the data source to work with by name

-h
    get the list of fields and supported arguments for the specified data source

-i=<index>
    specify the starting index for the result set

-m=<max>
    specify the maximum number of records to return

-s=<fields>
    specify the sorting to apply to any data being read from the data source

-o=<path>
    output the data from the data source to the specified file

-f=<text|csv|xml|html>
    the format to use when outputting the data from the data source

-c
    Whether to make string filter arguments case insensitive

```

To generate a report, specify the following:

- A data source
- An optional format
- Any parameters necessary to filter the data
- Any other output arguments

When fully constructed, a command to generate a report would look similar to this:

```
sudo mykonos-reports-cli -d=Datasource --format=true/false --parameter
```

Using the `-h` option in the CLI will bring up a help page with all of the available filters and parameters; a list of parameters is also available at the end of this document.

Supported arguments

The command line interface provides access to the raw information that the system uses. The following are the main arguments provided for accessing, outputting, and formatting the data returned by the CLI:

- **-d=<data source name>**: Define which data source to get data from (from the list of available data sources).
- **-h**: Get help documentation for the specified data source. This includes the accepted arguments and resulting fields.
- **-o=<path>**: Output data to a specified file (if not specified, then output is to the console).
- **-f=<text|csv|xml|html>**: The format to use when outputting the data from the data source, supported formats are: CSV, XML, HTML or text.
- **-l**: Get the list of all supported data sources and their names.
- **-i=<index>**: Specify the starting index for the result set.
- **-m=<max>**: Specify the maximum number of records to return.
- **-s=<fields>**: Specify the sorting to apply to any data being read from the data source.

Data Sources

The CLI provides users with the following data sources In order to access the data generated by the security engine and generate reports. The full list of Data sources can be viewed at any time with the **-l** option and a Data source can be loaded through the **-d=** argument:

- **Browser**: Data source that exposes information about known and detectable browsers.
- **Country**: Data source that exposes information about known countries.
- **Environment**: Data source that exposes information about environments.
- **Incident**: Data source that exposes information about incidents.
- **IncidentType**: Data source that exposes information about known incident types.
- **IpAddress**: Exposes IP Address information.
- **Location**: Data source that exposes information about locations.
- **OperatingSystem**: Data source that exposes information about known and detectable operating systems.
- **Profile**: Data source that exposes information about profiles.
- **Response**: Data source that exposes types of responses (counter attacks).
- **Session**: Data source that exposes information about sessions.

Formatting

All reports are generated with the default columns and sorting. You may reformat any report or change the sorting for the columns by supplying any optional filtering arguments when the report is generated. To generate a report that has all of the profile and session count data from the session's data source, include the following arguments in the command to add those columns:

```
sudo mykonos-reports-cli -d=Session --include-default=false --include-counts=true
```

```
mykonos@gal:~$ sudo /usr/sbin/mykonos-reports-cli -d=Session --include-default=true --include-counts=true
```

session.id	session.requestCount	session.errorCount	session.firstActive	session.lastActive	session.incidentCount	session.environmentCount	session.locationCount
660	1	1	3/25/11 2:24 PM	3/25/11 2:24 PM	null	1	1
662	8	1	3/25/11 2:24 PM	3/25/11 2:24 PM	2	1	1
668	1	0	3/25/11 2:47 PM	3/25/11 2:47 PM	1	1	1
674	1	0	3/25/11 2:47 PM	3/25/11 2:47 PM	1	1	1

Example Report

The CLI allows users to combine filtering arguments for data sources to create complex and detailed reports. These arguments follow the format:

```
--argumentName="Value"
```

For example, if a user wanted a report that displayed a list of all the sessions created after 09/12/2010, that also were from a client that used Windows XP, the following algorithm would be used:

```
sudo mykonos-reports-cli -d=Session --createdStartDate='09/12/2010 01:01 am' --environmentId=200
```


PART 3

Administration

- [General Tasks on page 85](#)
- [Verify on page 87](#)
- [EC2 Deployment on page 89](#)
- [Configuration Modes and Roles on page 97](#)
- [SRX Integration on page 107](#)
- [Appliance Management on page 109](#)
- [Spotlight Secure on page 119](#)
- [Security Monitor on page 123](#)
- [Autoresponse Defaults and Rule Creation on page 143](#)

General Tasks

- [Changing the Password on page 85](#)
- [Resetting the Password on page 85](#)

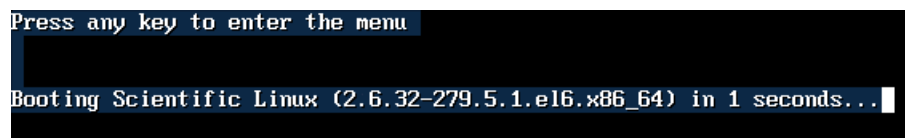
Changing the Password

The system password can only be changed from the underlying Linux command line. To do this, connect via the console or SSH. You will see the setup utility screen. Navigate to **Quit** to exit to the shell. Type **passwd** and follow the prompts.

Resetting the Password

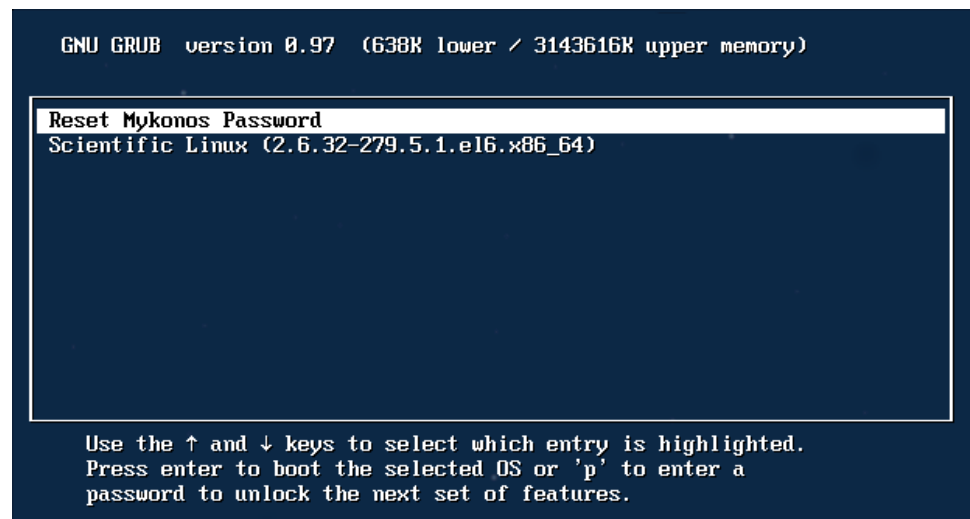
To reset the password, an appliance reboot is required. A boot menu option exists to reset the user credentials. By default, the appliance will boot normally, but by pressing any key before the operating system starts booting, you can get to the boot menu.

Figure 52: Boot Menu



Then you can select the option to reset the password.

Figure 53: Reset Password



CHAPTER 20

Verify

- [Verify the Installation on page 87](#)

Verify the Installation

In order to verify that your WebApp Secure appliance is processing traffic, use the following URL to access the appliance honeypot and confirm that it replies with a fake .htaccess.

http://<IP or Hostname>/.htaccess

The appliance should reply with something similar to the following. (Note that the actual fake .htaccess file may not look exactly like the example.)

```
<files "server_logs.txt">
  AuthUserFile /www/root/.htpasswd
  AuthType Basic
  AuthName "Error logs"
  Require valid-user
</files>
```

Your initial WebApp Secure configuration is complete. The appliance is ready to start protecting your applications.



NOTE: At this point, WebApp Secure is configured to secure one web server application.

CHAPTER 21

EC2 Deployment

- [EC2: Deploying Using the Command Line on page 89](#)
- [EC2: Deploying Using the Web Interface on page 90](#)
- [Assigning the Instance and IP Using the CLI on page 94](#)
- [Assigning the Instance and IP Using the Web Interface on page 94](#)
- [Verify the Instance is Running on page 95](#)

EC2: Deploying Using the Command Line

You must have an ec2tools environment setup prior to deploying the instance from the CLI. Please refer to the Amazon documentation for assistance in setting up your environment.

Next, make sure you have access to the image by entering: `./ec2-describe-images -x self -o "969756132034"`

The output should look similar to the following: **IMAGE ami-4d3df524 969756132034/Mykonos Appliance 969756132034** available If you do not see any output at all then you most-likely don't have access to our instance. Please contact Juniper Networks support to get help with this issue.

To deploy using the CLI, do the following:

1. Create a security group for the instance.
 - `./ec2-add-group Mykonos -d "Mykonos Appliance"`
 - `./ec2-authorize Mykonos -p 2022`
 - `./ec2-authorize Mykonos -p 80`
 - `./ec2-authorize Mykonos -p 443`
 - `./ec2-authorize Mykonos -p 5000`
 - `./ec2-authorize Mykonos -p 8080`



NOTE: It is recommended that you only allow ports 5000 and 8080 from known good IPs.

2. You can now deploy the new instance as follows:

- `./ec2-run-instances 'AMI ID' -k 'KEY PAIR' -t m1.large -g Mykonos`

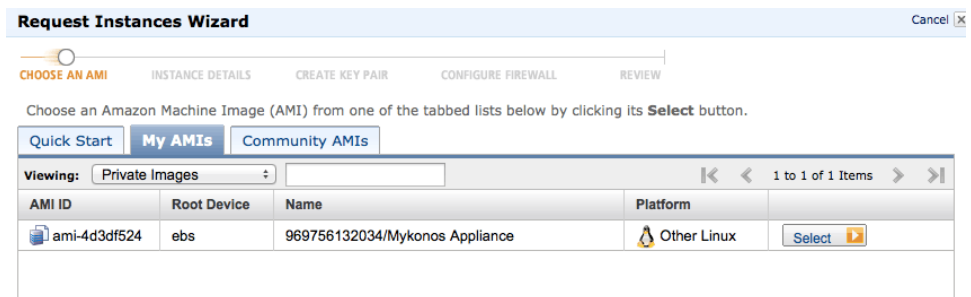
You must replace the AMI ID with the AMI ID listed in **Describe Image**. In this example, it would be **ami-4d3df524**. You must also replace **KEY PAIR** with the name of the key pair you are using to access the system.

EC2: Deploying Using the Web Interface

To deploy using the Web Interface, do the following:

1. Log into the AWS management console.
2. Select the **Amazon EC2** tab.
3. Click the **Launch Instance** button under the My Instances section.
4. Select **Launch Classic Wizard** then click **continue**.
5. Select the **My AMIs** tab.
6. Next to Viewing, select **Private Images**.
7. To the right of the appliance, click the **Select** button.

Figure 54: AWS management console



8. Change the Instance Type to **Large (m1.large)**.

Figure 55: Instance Type

Request Instances Wizard Cancel

CHOOSE AN AMI **INSTANCE DETAILS** CREATE KEY PAIR CONFIGURE FIREWALL REVIEW

Provide the details for your instance(s). You may also decide whether you want to launch your instances as "on-demand" or "spot" instances.

Number of Instances: 1 **Instance Type:** Large (m1.large, 7.5 GB)

Launch Instances

EC2 Instances let you pay for compute capacity by the hour with no long term commitments. This transforms what are commonly large fixed costs into much smaller variable costs.

Launch into:

EC2

Availability Zone: No Preference

Request Spot Instances

[< Back](#)
[Continue >](#)

- Click the **Continue** button. Then click the **Continue** button again on the next screen.

Figure 56: Configure Instance continued

Request Instances Wizard Cancel

CHOOSE AN AMI **CONFIGURE FIREWALL** REVIEW

Number of Instances: 1 **Availability Zone:** No Preference

Advanced Instance Options

Here you can choose a specific **kernel** or **RAM disk** to use with your instances. You can also choose to enable CloudWatch Detailed Monitoring or enter data that will be available from your instances once they launch.

Kernel ID: Use Default **RAM Disk ID:** Use Default

Monitoring: ☐ Enable CloudWatch detailed monitoring for this instance (additional charges will apply)

User Data:

☒ as text

☐ as file ☐ base64 encoded

Termination Protection: ☐ Prevention against accidental termination.

Shutdown Behavior: Stop Choose the behavior when the instance is shutdown from within the instance.

[< Back](#) [Continue >](#)

- Add a **Name** to make the instance easily recognizable and click **Continue**.

Figure 57: Instance, Configure Name

Request Instances Wizard Cancel

CHOOSE AN AMI **INSTANCE DETAILS** CREATE KEY PAIR CONFIGURE FIREWALL REVIEW

Add tags to your instance to simplify the administration of your EC2 infrastructure. A form of metadata, tags consist of a case-sensitive key/value pair, are stored in the cloud and are private to your account. You can create user-friendly names that help you organize, search, and browse your resources. For example, you could define a tag with key = Name and value = Webserver. You can add up to 10 unique keys to each instance along with an optional value for each key. For more information, go to [Using Tags](#) in the *EC2 User Guide*.

Key (127 characters maximum)	Value (255 characters maximum)	Remove
Name	Mykonos	

[Add another Tag.](#) (Maximum of 10)

[Back](#)[Continue](#)

11. Select or create your **Key Pair** and click **Continue**.

Figure 58: Instance, Create Key Pair

Request Instances Wizard Cancel

CHOOSE AN AMI **INSTANCE DETAILS** **CREATE KEY PAIR** CONFIGURE FIREWALL REVIEW

Public/private key pairs allow you to securely connect to your instance after it launches. To create a key pair, enter a name and click **Create & Download your Key Pair**. You will then be prompted to save the private key to your computer. Note, you only need to generate a key pair once - not each time you want to deploy an Amazon EC2 instance.

☒ **Choose from your existing Key Pairs**

Your existing Key Pairs*: Main

☐ **Create a new Key Pair**

☐ **Proceed without a Key Pair**

[Back](#)[Continue](#)

12. Select **Create a New Security Group** and enter the following information:

- Group Name: Mykonos
- Group Description: Mykonos Appliance

- Port Range: 2022
- Port Range: 80
 - Add Rule
- Port Range: 443
 - Add Rule
- Port Range: 5000
 - Add Rule

Figure 59: Instance Create Security Group

☐ Choose one or more of your existing Security Groups

☒ Create a new Security Group

Group Name

Group Description

Inbound Rules


Create a new rule:

Port range:

(e.g., 80 or 49152-65535)

Source:

(e.g., 192.168.2.0/24, sg-47ad482e, or 1234567890/default)

 Add Rule

TCP	Port (Service)	Source	Action
	2022	0.0.0.0/0	Delete
	80 (HTTP)	0.0.0.0/0	Delete
	443 (HTTPS)	0.0.0.0/0	Delete
	5000	0.0.0.0/0	Delete
	8080 (HTTP*)	0.0.0.0/0	Delete

[< Back](#)
[Continue >](#)

- Click the **Continue** button.
- Click the **Launch** button.

Figure 60: Instance, Review and Launch

Request Instances Wizard Cancel

CHOOSE AN AMI INSTANCE DETAILS CREATE KEY PAIR CONFIGURE FIREWALL **REVIEW**

Please review the information below, then click **Launch**.

AMI: Other Linux AMI ID ami-4d3df524 (x86_64) [Edit AMI](#)

Number of Instances: 1

Availability Zone: No Preference

Instance Type: Large (m1.large)

Instance Class: On Demand [Edit Instance Details](#)

Monitoring: Disabled **Termination Protection:** Disabled

Tenancy: Default

Kernel ID: Use Default **Shutdown Behavior:** Stop

RAM Disk ID: Use Default

User Data: [Edit Advanced Details](#)

Key Pair Name: Main [Edit Key Pair](#)

Security Group(s): sg-7ae70112 [Edit Firewall](#)

[Back](#) [Launch](#)

Assigning the Instance and IP Using the CLI

1. Request a new public IP address: `./ec2-allocate-address` Note the IP it returns.
2. Get the Instance ID of the WebApp Secure Instance: `./ec2-describe-instances` Locate the Instance ID of the Appliance.
3. Now you can associate the IP with the WebApp Instance: `./ec2-associate-address IP -i 'Instance ID'`

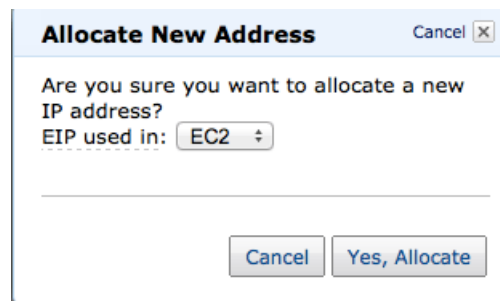


NOTE: Replace the 'IP' with your IP address and the 'Instance ID' with the ID of your instance.

Assigning the Instance and IP Using the Web Interface

1. Select the **Amazon ec2** tab.
2. Select **Elastic IPs** under **Navigation** on the left.
3. Click the **Allocate New Address** button under the **Address** section.
4. Click **Yes, Allocate**.

Figure 61: Allocate New Address



5. Click **Associate Address**.
6. Select the **Appliance**.
7. Click **Yes, Associate**.

Verify the Instance is Running

- Purpose** At this point, your instance should be up and running. You can use the web interface or the CLI tools to verify this.
- Action** To access your instance, you must have a copy of the key pair you used for the instance.
`ssh -i 'PATH TO KEY PAIR' mykonos@'IP' -p 2022`
- You should be granted access, and the TUI will be launched for you. From there, you can configure the appliance.

CHAPTER 22

Configuration Modes and Roles

- [Basic Configuration Mode on page 97](#)
- [Expert Configuration Mode on page 98](#)
- [Role-Based Administrator Access Control on page 99](#)
- [Configuring Role-Based Access Control on page 99](#)
- [RBAC Groups and Roles on page 100](#)
- [Edit User Preferences on page 103](#)
- [Unblock Login Ban on page 104](#)

Basic Configuration Mode

By default, any configuration page navigated to will result in the Basic Configuration page for that particular section. You can view the various sections of Configuration underneath the **Configuration** page on the left side navigation. The available sections are as follows:

- **Security Engine:** Core Security Engine options, such as health checks, and whitelisting.
- **Processors:** Security Processors are pluggable modules that process HTTP traffic and perform actions.
- **Services:** Services run in the background, performing tasks such as sending alerts, generating reports, or performing maintenance tasks.
- **Proxy / Backends:** Core proxy settings, such as backend servers and SSL.
- **Applications:** By default, the system will secure only one application. Adding multiple profiles will enable you to protect multiple applications with their own separate settings.
- **Backups:** Configure backup frequency, retention, and pushes to FTP or SSH servers.
- **Logging:** Options for logging access on the management interfaces, as well as logging the various security incidents triggered by WebApp Secure.
- **Response Rules:** Configure how the system responds to threats, or create custom response rules.
- **Licensing:** Add or update licensing information to ensure operation of your system.
- **Users and Groups:** Add or update user roles and permissions.

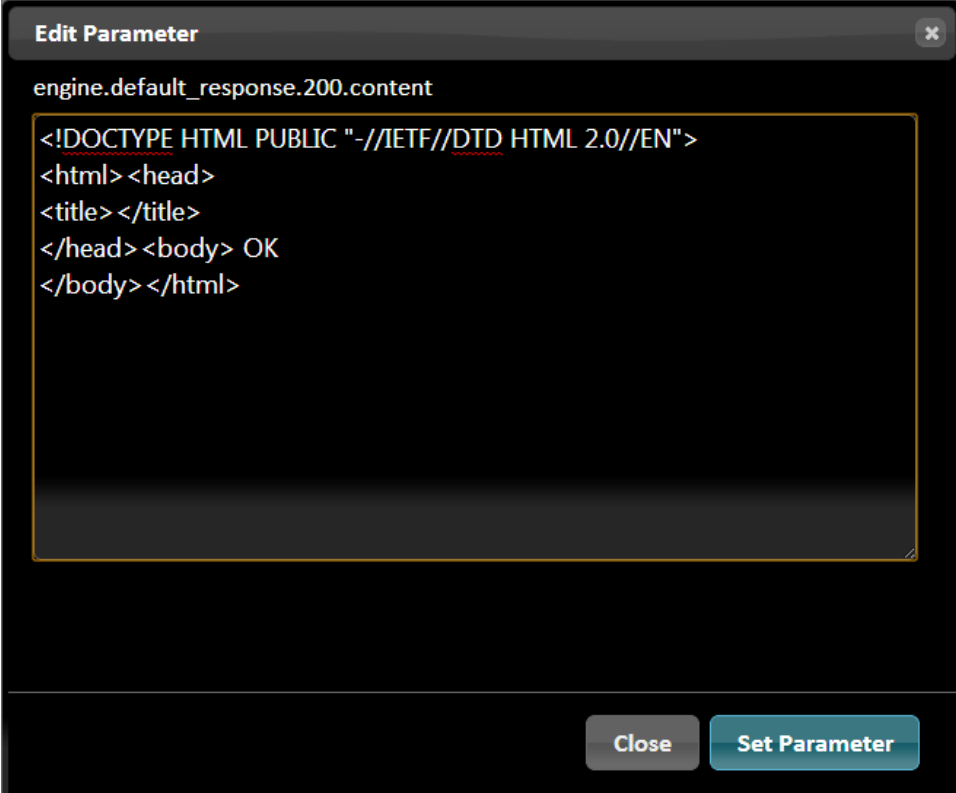
Expert Configuration Mode

In most cases, using the standard configuration interface should be sufficient. Some users might prefer editing the configuration parameters at the key-value level. Expert Mode is a way to view all the configuration attributes from the Web UI. You can reach Expert Mode by clicking the **Expert Mode** button on the upper right side of the Configuration page.

To edit any configuration parameter, first navigate to the correct parameter name. The table is ordered alphabetically, and you can browse through the help documentation for various parameters by using the **help** keyword in the CLI (mykonos-shell).

Once you find the entry, you can edit it, remove it, or reset it to its default value using the icons on the left side of the table. When editing a parameter, you are given a text box in which to make the edit. Some parameters are Base64 encoded (like HTML responses), but will be displayed in an un-encoded form. Make your changes and click **Set Parameter** to save the changes.

Figure 62: Edit Parameter



Edit Parameter

engine.default_response.200.content

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title></title>
</head><body> OK
</body></html>
```

Close Set Parameter



WARNING: Even after you save the parameter, the changes to configuration have not been saved and set until you click the Save button at the bottom of

the page. If you navigate away from the page before saving the entire configuration, parameters are not saved.

Role-Based Administrator Access Control

Role-Based Access Control (RBAC) is a way to assign different levels of administrator functionality to different users. You can assign roles to various users that exist on a configured LDAP or RADIUS server. The first step in integrating with your existing LDAP or RADIUS service is to give WebApp Secure the connection information. In the web UI, navigate to 'Configuration >> Users and Groups' and click on 'Manage Authentication Settings'. On the resulting page, input all information relating to your LDAP or RADIUS server and click 'Save'. You should now see the corresponding service as "Enabled" under the Authentication section of Users and Groups. Once the server has been connected to JWAS, the next step is to configure roles for various users. By default, the user "mykonos" is enabled and given the role "Super Administrator". To add additional users, click the 'Add User' link. You will be prompted to enter a Username and will be given a choice of which groups you want that user to inherit. A complete description of all roles is available by clicking on 'View Role Descriptions' underneath the Roles dropdown. A more simplistic table of roles and their corresponding permissions are given in Appendix D, RBAC Groups and Roles.

Configuring Role-Based Access Control

1. In the Web UI, go to **Configuration > Users and Groups**.
2. Click **Manage Authentication Settings**.
3. Enter all information relating to your LDAP or RADIUS server and click **Save**. You should now see the corresponding service as Enabled under the Authentication section of Users and Groups.
4. The next step is to configure roles for various users. By default, the user *mykonos* is enabled and given the role *Super Administrator*. To add additional users, click the **Add User** link.
5. You are prompted to enter a **Username** and you are given a choice of which groups you want the user to inherit. A complete description of all roles is available by clicking **View Role Descriptions** beneath the **Roles** drop down list. A more simplistic table of roles and their corresponding permissions can be found in Appendix D, RBAC Groups and Roles.

Figure 63: Users and Groups, Add User

Figure 64: Assigned Roles

Username	Assigned Role(s)	Actions
johndoe	RBAC Administrator (System Group: rbacadmin), Web UI Administrator (System Group: webuiadmin)	
mykonos	Super Administrator (System Group: superadmin)	

Add User



NOTE: Because WebApp Secure doesn't actually create users on the appliance itself but merely maps the username to the given permissions, the only way to effectively remove the user is to strip them from all roles. After removing roles and saving, the entry in the Authorization table is removed.



NOTE: WebApp Secure doesn't allow the last RBAC Administrator role to be deleted. It is possible to remove your own permissions, though, essentially locking you out of the system. Similarly, re-initializing the configuration settings will wipe out all user-role mappings, and the mykonos user will be the only one able to assign roles.



NOTE: Any violations of access control (a user trying to access some part of the system they aren't configured to access) will be logged to the audit log.

RBAC Groups and Roles

This is a list of all WebApp Secure roles, and their corresponding permissions.

Table 6: RBAC Groups and Roles.

	Super Administrator	Security Administrator	Security Support Staff	RBAC Administrator	Web UI Administrator	Device Administrator	Security User
Can Manage Processors	Yes	Yes	No	No	Yes	No	No
Can Manage Response Rules	Yes	Yes	No	No	Yes	No	No
Can View System Status	Yes	Yes	Yes	No	Yes	Yes	Yes
Can Edit Profiles	Yes	Yes	No	No	Yes	No	Yes
Can Use Expert Mode	Yes	No	No	No	Yes	No	No
Can Delete Profiles	Yes	Yes	No	No	Yes	No	Yes
Can Manage Logical Services	Yes	Yes	No	No	Yes	Yes	No
Can View Security Data	Yes	Yes	Yes	No	Yes	No	Yes
Can Manage Licensing	Yes	No	No	No	Yes	Yes	No
Can Manage Authentication	Yes	No	No	No	Yes	Yes	No
Can Manage Applications	Yes	No	No	No	Yes	No	No
Can Import Configuration	Yes	No	No	No	Yes	No	No
Can Initialize Appliance	Yes	No	No	No	No	Yes	No
Can Log Into Web UI	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Can Manage Backups	Yes	No	No	No	Yes	Yes	No

Table 6: RBAC Groups and Roles. *(continued)*

	Super Administrator	Security Administrator	Security Support Staff	RBAC Administrator	Web UI Administrator	Device Administrator	Security User
Can Activate Responses	Yes	Yes	No	No	Yes	No	Yes
Can Export Configuration	Yes	No	No	No	Yes	No	No
Can Manage Physical Services	Yes	No	No	No	No	Yes	No
Can Manage Logging	Yes	No	No	No	Yes	Yes	No
Can Manage Spotlight	Yes	Yes	No	No	Yes	No	No
Can Deactivate Responses	Yes	Yes	No	No	Yes	No	Yes
Can Log Into Console	Yes	No	No	No	No	Yes	No
Can Schedule Reports	Yes	Yes	No	No	Yes	No	Yes
Can Update Appliance	Yes	No	No	No	Yes	Yes	No
Can Manage High Availability	Yes	No	No	No	Yes	Yes	No
Can Configure Updates	Yes	No	No	No	No	Yes	No
Can Manage Security Engine	Yes	Yes	No	No	Yes	No	No
Can Run Reports	Yes	Yes	Yes	No	Yes	No	Yes
Can Manage Authorization	Yes	No	No	Yes	Yes	No	No

Table 6: RBAC Groups and Roles. *(continued)*

	Super Administrator	Security Administrator	Security Support Staff	RBAC Administrator	Web UI Administrator	Device Administrator	Security User
Can Manage SRX Settings	Yes	No	No	No	No	Yes	No
Can Restart Appliance	Yes	No	No	No	No	Yes	No

Edit User Preferences

User Preferences control the appearance of the user interface and how certain information is displayed. Click the **Edit Preferences** link at the top right of the UI to access the User Preferences screen. The following preference settings are available:

- **Skin:** Change the color and overall look of the UI.
- **Language:** At this time, only English is supported.
- **Timezone:** Change the timezone setting. Note that this field defaults to UTC.
- **Prompt Level:** Change the amount of help text displayed for each field. If you are familiar with the product, you may prefer abbreviated help text to lessen the amount of information on the screen.
- **Spotlight Name Preference:** If Spotlight is enabled, you can select to have Spotlight global names displayed in attacker lists and reports. You can also choose to display only local names or to display both local and global names.
- **Auto Refresh:** You can enable or disable this setting. Note that Auto refresh affects all security related screens, including the dashboard, lists of hackers, sessions, locations, and incidents.
- **Refresh Interval:** Change the refresh interval. The minimum value you can set here is 10 seconds.
- **Records Per Page:** Change the number of records to display on a per page basis.
- **Debug Mode:** You can enable or disable this setting. Certain UI items are hidden by default. For debugging purposes, you can enable this checkbox to reveal all hidden items.

Figure 65: User Preferences

User Preferences

Skin	Dark (Default)	To accommodate aesthetic preferences, you may select an alternate skin.
Language	English (US)	Where available, we will use localized text throughout the system.
Timezone	UTC	By default, items in the User Interface are shown in UTC. To change this, set this value to your preferred timezone.
Prompt Level	Abbreviated	By default, help text is shown inline. If you are familiar with the product, you may wish to have an abbreviated prompt level.
Spotlight Name Preference	Display Global Names If Available	If spotlight is enabled, we will use this setting to determine how to display attacker profile names to you, throughout the UI.
Auto Refresh	<input checked="" type="checkbox"/>	Enable auto refreshing of datasets displayed on pages. (Requires JavaScript)
Refresh Interval	300	How often, in seconds, automatic refreshes will occur, if 'auto refresh' is enabled.
Records Per Page	15	The default number of records returned per page. Used for all lists of data throughout the entire UI.
Debug Mode	<input type="checkbox"/>	Certain items in the user interface can be usable for troubleshooting, but are hidden by default. To see them, set this value.

Save

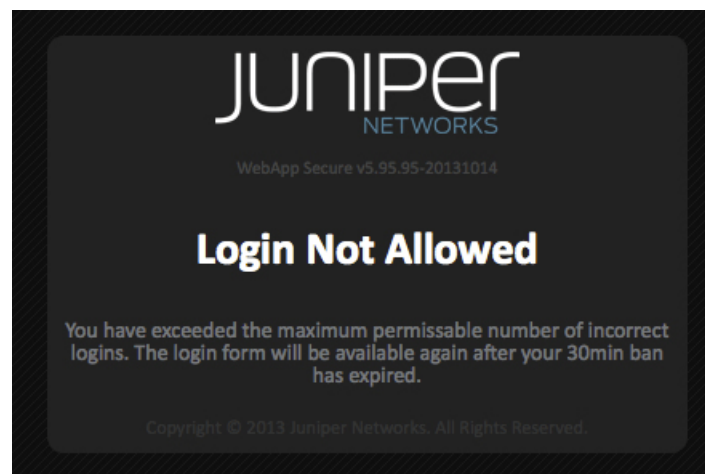
Unblock Login Ban

After six failed login attempts, the IP address in question is blocked from further login attempts for a period of thirty minutes.



NOTE: Once you reach four failed login attempts, you are warned that you will be banned after two more failed attempts.

Figure 66: Blocked Login



A system administrator can remove this block before the thirty minute time-frame has expired by using the following command:

```
[root@johnsmith-vm mykui]# mykonos-shell
Welcome to the WebApp Secure CLI
> system
system> unlock_webui
Confirm Unlock [y|N]: y
system> exit
```



NOTE: This command unlocks all locked accounts.

SRX Integration

- [Testing the SRX Integration Configuration on page 107](#)

Testing the SRX Integration Configuration

Purpose To verify the configuration of the WebApp Secure portion of the SRX integration, do the following

- Action**
1. Create a profile by accessing the .htaccess file (explained in “Verify the Installation”).
 2. Navigate to the WebApp Secure web interface and find the newly created profile.
 3. Manually activate the **Filter on SRX** Counter Response.
 4. Log into the SRX CLI, and run the command **show configuration firewall** (or **show firewall** if in).

You should see a new filter created with the name you gave in configuration, and a new term within that filter called that you also named within configuration. It should appear similar to the following (depending on how you set up your filter and actions):

```
family inet {
  filter my_filter {
    term block {
      from {
        address {
          10.10.10.10/32;
        }
      }
      then {
        reject;
      }
    }
    term default {
      then {
        accept;
      }
    }
  }
}
```

In the example above, 10.10.10.10 is the IP of the profile you activated the Counter Response on. This is telling the SRX to reject the IP of the profile at the gateway level.

Note the default term below the block term which will act as an accept-all in the case that the block term's action has been changed to **next term**.

You can also verify the line with the IP address gets deleted when deactivating the Counter Response.



.....

NOTE: When there are no IPs to block, the SRX defaults to * or All Traffic. This would effectively block all traffic from that interface! To counter this, WebApp Secure changes the action from your configured entry to next term, essentially letting the next term within the filter deal with the traffic. Because you set up a default term to handle this case (see Configuration), the next term simply accepts all traffic.

.....

This filter should now look as follows:

```
family inet {
  filter my_filter {
    term block {
      then next term;
    }
    term default {
      then {
        accept;
      }
    }
  }
}
```

This is indicating that all traffic will be sent through this term, but the action is simply passing the packet onto the next term in the filter, which is our default term that will accept all traffic.

Appliance Management

- [Restart and Shutdown the Appliance on page 109](#)
- [System Updates on page 109](#)
- [Statistics on page 112](#)
- [Master - Slave Mode on page 115](#)
- [Managing Services on page 115](#)
- [Backup and Recovery Overview on page 115](#)
- [Health Check URL on page 116](#)
- [Backup and Recovery Overview on page 117](#)
- [Restoring a Backup on page 117](#)

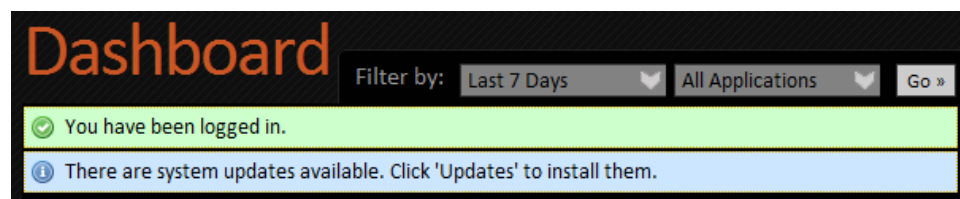
Restart and Shutdown the Appliance

To safely shutdown or reboot the appliance, SSH into the appliance and invoke the mykonos-shell by entering **system shutdown** or **system reboot**.

System Updates

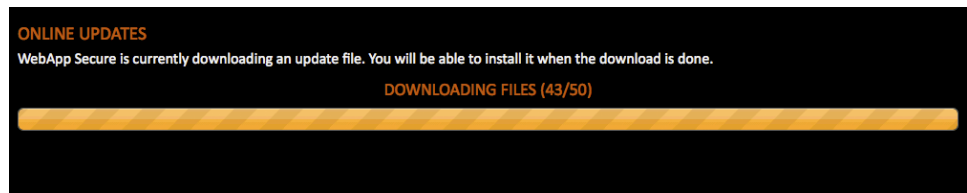
Provided WebApp Secure has Internet access, either direct or through a configured proxy, it will automatically check for software updates every night and download them when new ones are available. However, the appliance will not automatically apply updates. For security and stability, an administrator must manually apply updates. The UI informs the you that there is an update by a banner indicator at the top of the page.

Figure 67: Dashboard, Updates



While WebApp Secure checks for updates every night, you can force the appliance to check for updates at any time by clicking the **check for updates** link under **Online Updates**. WebApp Secure will fetch any available online updates at this time. You can see progress of the download via a status bar.

Figure 68: Downloading Update



WebApp Secure can also upload updates manually, without an internet connection. After uploading the package to the appliance (via the Web UI's **Updates** page), it will become available to the updates system, and you will be able to apply the update as described here.



WARNING: While WebApp Secure is uploading offline updates, you should stay on the Updates pane until the upload is complete.

If an update is available (either an uploaded offline update or an automatically downloaded one), you can view the available update package along with any information about it, including the package name, version, whether or not a reboot is required after installing the update, a description, and list of changes. After reviewing the changes you can choose to apply the update by clicking the **Update Selected** button at the bottom of the package table.

Figure 69: Update Description

ONLINE UPDATES

[check for updates](#)

You have 1 update available.

Update?	Name	Version	Reboot (?)
<input checked="" type="checkbox"/>	Junos WebApp Secure Core	4.2.2-96	No

DESCRIPTION

Upscaling the resurgent networking exchange solutions, achieving a breakaway systemic electronic data interchange system synchronization, thereby exploiting technical environments for mission critical broad based capacity constrained systems. Fundamentally transforming well designed actionable information whose semantic content is virtually null. Empowerment in information design literacy demands the immediate and complete disregard of the entire contents of this cyberspace communication.

CHANGELOG

9/26/2012
User Guide updated with section explaining offline vs. online updates.

9/23/2012
Administrator can now configure the maximum login attempts on a page before forcing a captcha response.

9/19/2012
Reports system overhauled. Reports can now be exported from many pages in the security monitor.

The system will update and inform the user of its progress via a status bar.

Figure 70: Updating the Application

Update process started.

Your updates are in progress. Please do not leave this page until the update process has completed.

Updating Junos WebApp Secure

RESTARTING SERVICES...



NOTE: At this time, it is not possible to roll back to earlier versions of the appliance software.

Statistics

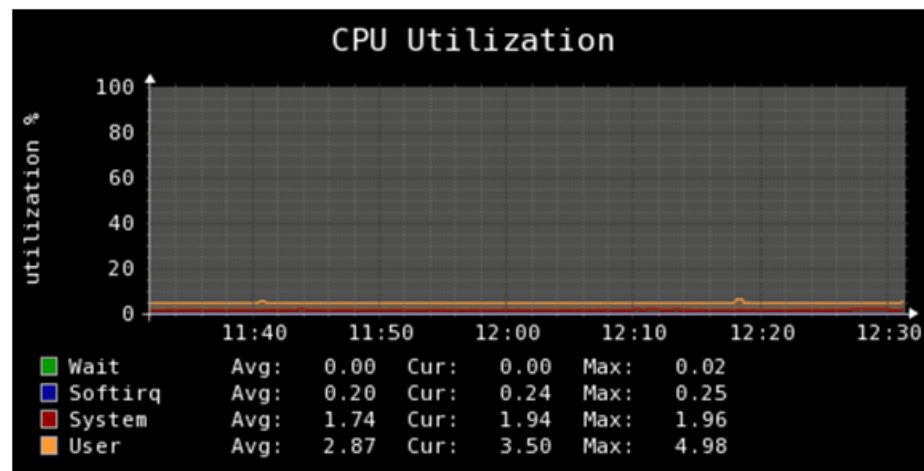
WebApp Secure software allows for standard SNMP system monitoring. All statistics available on a typical Linux system would be available to WebApp Secure through standard system SNMP mibs. In addition, WebApp Secure currently offers six types of systems statistics in a form of graphs. They include CPU Utilization, CPU Load Average, Memory Utilization, Network Traffic, Proxy Connection and Proxy Requests. They can be accessed via **System Status** button in the top menu of the Configuration management interface. Depending on the desired level of details, the statistics can be viewed for the Last Hour, Last 12 Hours, Last Day, Last Week and, finally, Last Month (always last 30 days).

Below are the details of the statistics that are available for each type:

CPU Utilization

- **Wait** - Percentage of CPU time spent in wait (on disk)
- **Softirq** - Percentage of CPU time spent handling software interrupts
- **System** - Percentage of CPU time spent in kernel space
- **User** - Percentage of CPU time spent in user space

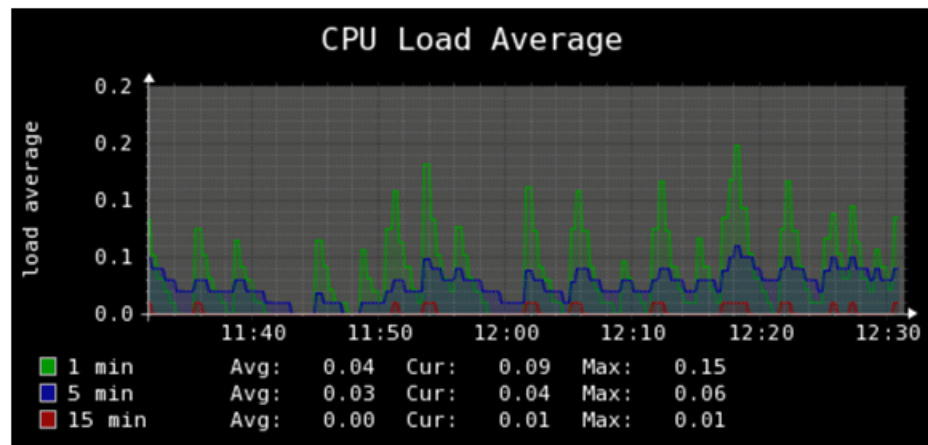
Figure 71: CPU Utilization



CPU Load Average

- 1 min - CPU Load for the last minute
- 10 min - CPU Load for the last 10 minutes
- 15 min - CPU Load for the last 15 minutes

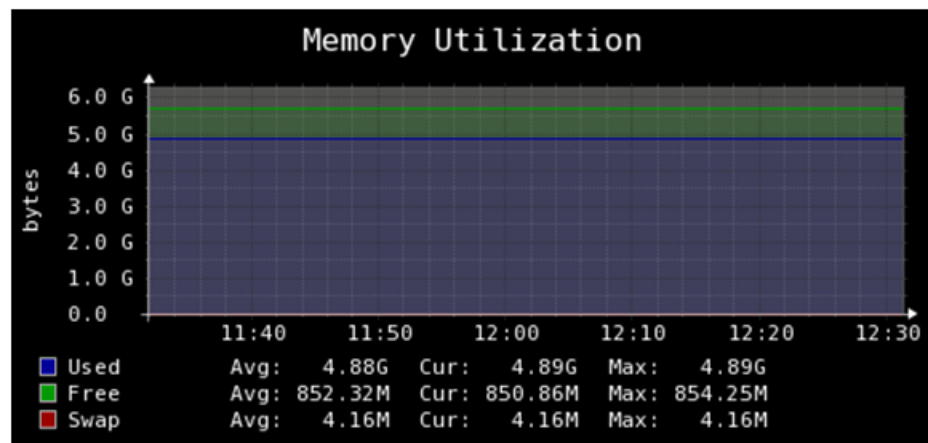
Figure 72: CPU Load Average



Memory Utilization

- Used - Amount of memory used
- Free - Amount of memory free
- Free - Amount of memory free

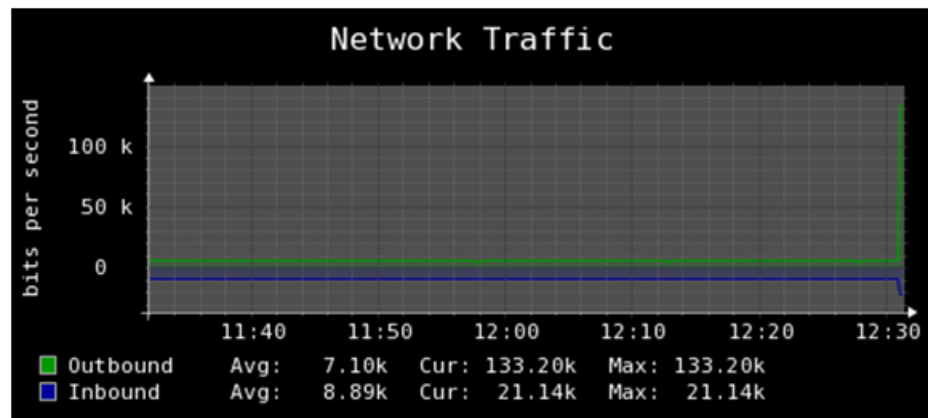
Figure 73: Memory Utilization



Network Traffic

- Outbound - Amount of traffic leaving the box
- Inbound - Amount of traffic entering the box

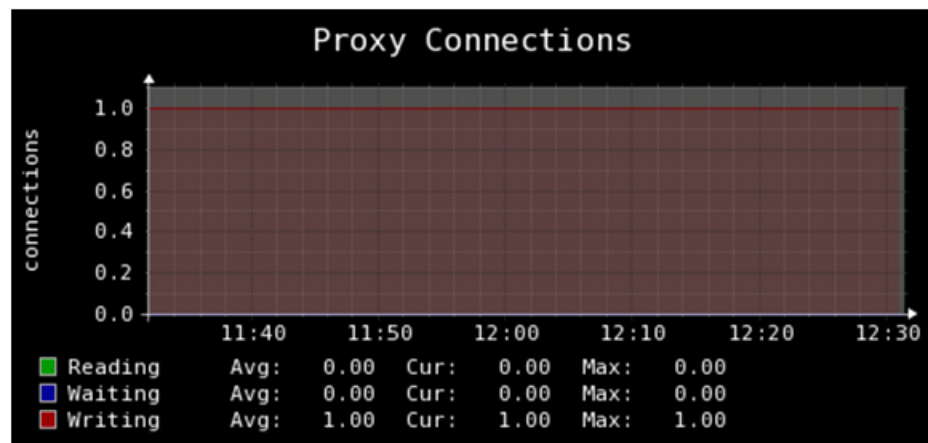
Figure 74: Network Traffic



Proxy Connections

- Reading - Number of TCP connections reading data
- Waiting - Number of TCP connections waiting
- Writing - Number of TCP connection writing Proxy

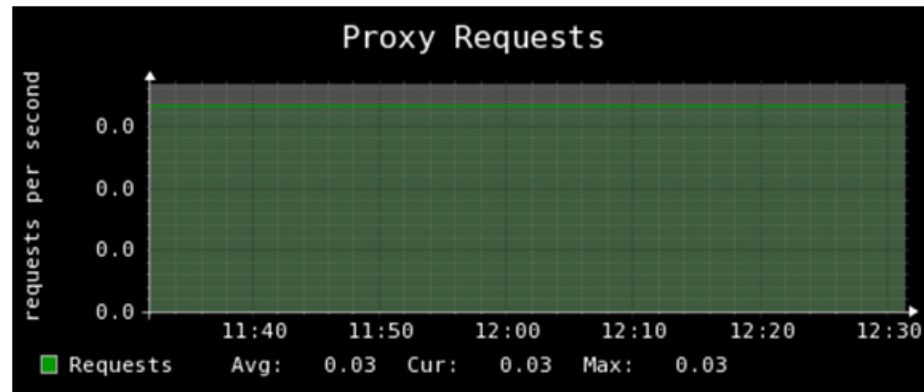
Figure 75: Proxy Connections



Proxy Requests

- Requests - Current number of HTTP/HTTPS requests being processed

Figure 76: Proxy Requests



Master - Slave Mode

In the case of the appliance running in multi-server mode, the systems statistics will show details for each node in the cluster as well as the key cumulative data across the entire cluster. Each system will be presented as a tab in the Configuration UI's system status page with the **Aggregate** tab being first. The **Aggregate** tab always shows CPU Utilization, Network traffic, Proxy connections and Proxy requests collected from the entire active WebApp Secure cluster.

Managing Services

WebApp Secure runs on top of an optimized, hardened, Linux installation. The core of WebApp Secure is several programs that run as services or daemons that work together to defend your web applications. The services context in the CLI lets you check the status of WebApp Secure services, or start, stop, or restart, them, if necessary. Note that these are all console functions, and not accessed through the Web interface.

Backup and Recovery Overview

The administrator can adjust the backup settings in the Web UI's configuration by navigating to **Configuration > Backups**. WebApp Secure stores its backups in the `/home/mykonos/backups` directory.

Figure 77: Backup Configuration

You can invoke a backup from the command line mykonos-shell, by entering **system backup**. You will be prompted to confirm, and a file will be created in **/home/mykonos/backups/**.



NOTE: The file will be named **mykonos-<version>-<hostname>-<timestamp>.myk**

Health Check URL

The Health Check URL lets an external system (typically a load balancer) that confirms the WebApp Secure system is operating properly. The system will generate a file name consisting of an arbitrary string of characters; make a note of it. If an HTTP request is sent to WebApp Secure for this file name, it will return **200 OK**, with a code in the body of the message. The responses are as follows.

Table 7: Health Check responses and corresponding meanings.

Response	Meaning
No response	WebApp Secure is offline
200 OK, plus OK	WebApp Secure is fully functional and is protecting your web sites
200 OK, plus DISABLED	WebApp Secure is running, but has been disabled or the license has expired
200 OK, plus STAND-BY [...]	WebApp Secure is waiting on an external resource. The contents of [...] will provide additional information

The format of the HTTP request should be: **http://jws_fullyqualifieddomainname_or_IPaddress/filenamegeneratedbyjws**

Backup and Recovery Overview

The administrator can adjust the backup settings in the Web UI's configuration by navigating to **Configuration > Backups**. WebApp Secure stores its backups in the `/home/mykonos/backups` directory.

Figure 78: Backup Configuration

You can invoke a backup from the command line mykonos-shell, by entering **system backup**. You will be prompted to confirm, and a file will be created in `/home/mykonos/backups/`.



NOTE: The file will be named `mykonos-<version>-<hostname>-<timestamp>.myk`

Restoring a Backup

To restore the data that is displayed in the Monitoring Console from a back up, you must use the command line utility specialized for the database backups. This does not include configuration or other system settings, only database information.



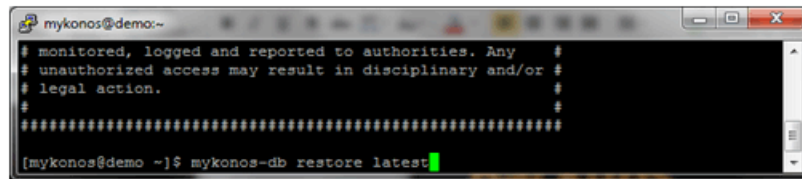
NOTE: If the data is being restored to the console, a database backup will need to be specified from `/usr/share/msa/database` or use the `latest` option to restore from the last valid backup.

A restore is run with the following command: `sudo mykonos-db <option>`

The options for the command above are as follows:

- backup
- restore (filename)
- restore latest
- clean

Figure 79: Restore Backup



CHAPTER 25

Spotlight Secure

- Enabling Spotlight Secure on page 119

Enabling Spotlight Secure

To enable the Spotlight Secure service on your appliance, do the following:

1. Navigate to **Spotlight** on the left side navigation menu in the WebApp Secure Web UI.
2. Click the **Enable** button in the top right corner of the page.

You can optionally choose to customize some of the submission and resolution intervals (using the **Configure** button), but for most applications the defaults will be fine.

Figure 80: Spotlight Secure, Enable

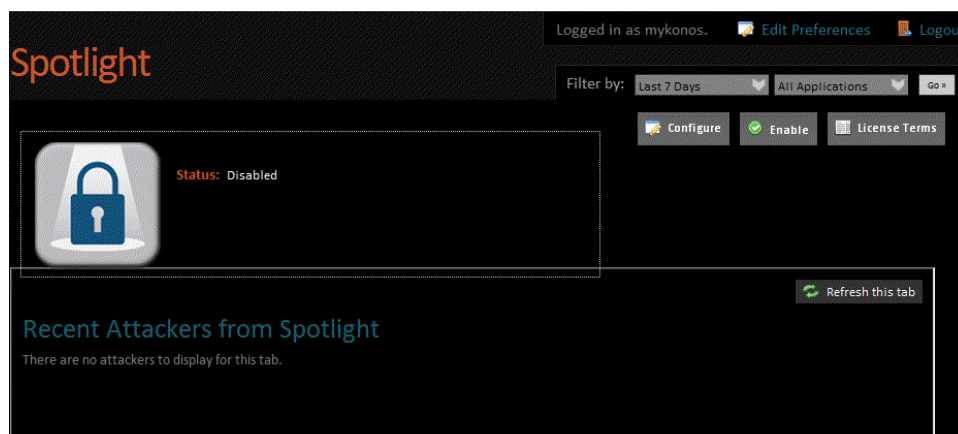
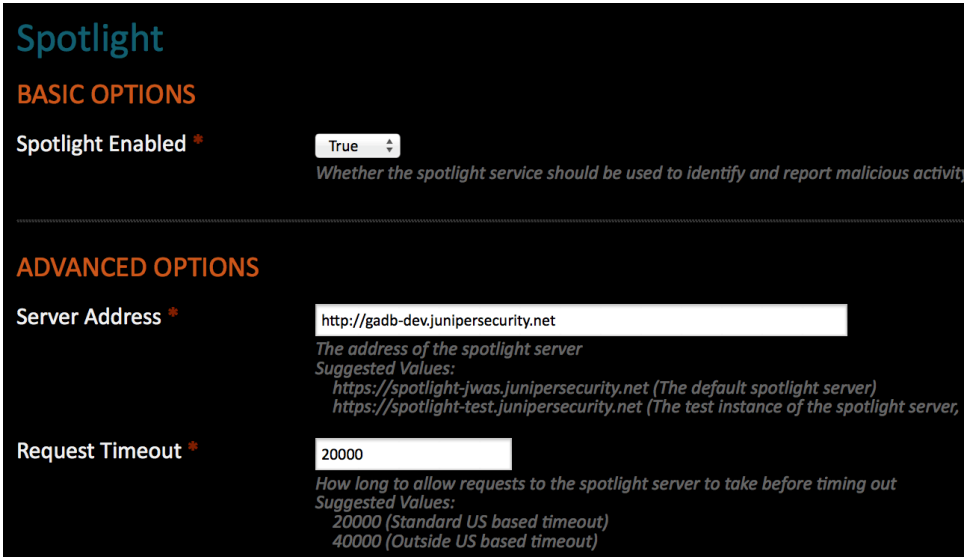


Figure 81: Spotlight Secure Configuration Screen



Spotlight

BASIC OPTIONS

Spotlight Enabled * ☐ True
Whether the spotlight service should be used to identify and report malicious activity

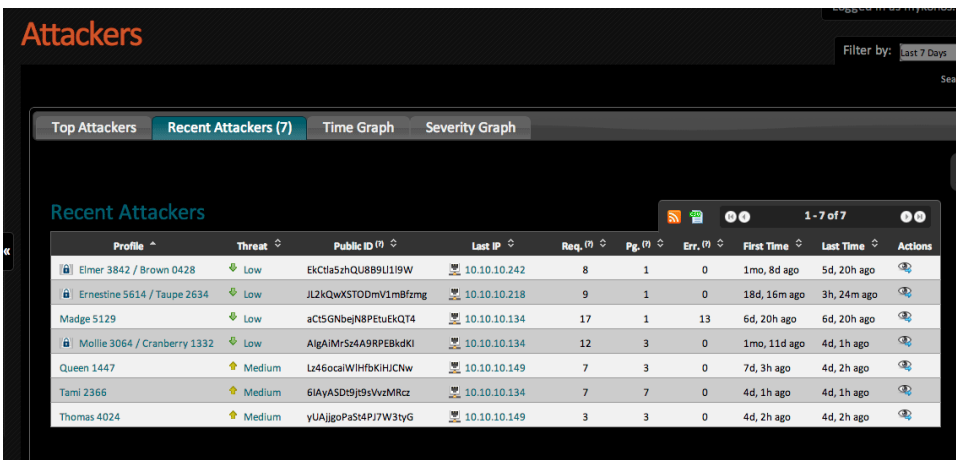
ADVANCED OPTIONS

Server Address *
The address of the spotlight server
 Suggested Values:
<https://spotlight-jwas.junipersecurity.net> (The default spotlight server)
<https://spotlight-test.junipersecurity.net> (The test instance of the spotlight server,

Request Timeout *
How long to allow requests to the spotlight server to take before timing out
 Suggested Values:
 20000 (Standard US based timeout)
 40000 (Outside US based timeout)

Once an attacker from another site visits a page on your site, a Spotlight Profile will be created for that user. Unlike local Profiles, Spotlight Profiles aren't automatically considered malicious -- They haven't harmed your site yet. Having attackers from other sites consolidated on the Spotlight page in the UI does allow you to keep close tabs on them. You can view the Spotlight profiles from the Spotlight page. Each Spotlight profile will be displayed in a row, with information such as their Local Profile name, Global (Spotlight) Profile name, and the first and last times seen both locally and globally.

Figure 82: Recent Attackers: Global and Local Names



Attackers

Filter by: Last 7 Days

Top Attackers Recent Attackers (7) Time Graph Severity Graph

Recent Attackers

Profile	Threat	Public ID	Last IP	Req.	Pg.	Err.	First Time	Last Time	Actions
Elmer 3842 / Brown 0428	Low	EkCtla5zhQU8B9U1I9W	10.10.10.242	8	1	0	1mo, 8d ago	5d, 20h ago	
Ernestine 5614 / Taupe 2634	Low	JL2kQwXSTODmV1m8fmg	10.10.10.218	9	1	0	18d, 16m ago	3h, 24m ago	
Madge 5129	Low	aCt5GNbejN8PEtuEKQ74	10.10.10.134	17	1	13	6d, 20h ago	6d, 20h ago	
Mollie 3064 / Cranberry 1332	Low	AlgAlMr5z4A9RPEBkdKl	10.10.10.134	12	3	0	1mo, 11d ago	4d, 1h ago	
Queen 1447	Medium	Lz46ocalWIHfbKHJCNw	10.10.10.149	7	3	0	7d, 3h ago	4d, 2h ago	
Tami 2366	Medium	6IAyASD9j9sVzMRcz	10.10.10.134	7	7	0	4d, 1h ago	4d, 1h ago	
Thomas 4024	Medium	yUAJgoPa5t4Pj7W3tyG	10.10.10.149	3	3	0	4d, 2h ago	4d, 2h ago	

You can view the Spotlight attackers' activities on your system on the **Sessions and Attackers** page. They are displayed with the same information as local attackers, and are indicated by the Spotlight icon next to their name.






















Figure 83: Recent Attackers: Global Names

Attackers

Filter by:

Top Attackers Recent Attackers (7) Time Graph Severity Graph

Recent Attackers 1 - 7 of 7

Profile ^	Threat ^	Public ID ^	Last IP ^	Req. ^	Pg. ^	Err. ^	First Time ^	Last Time ^	Actions
 Brown 0428	Low	EkCtla5zhQU889U119W	 10.10.10.242	8	1	0	1mo, 8d ago	5d, 20h ago	
 Cranberry 1332	Low	AlgAlMrS4A9RPEBkdKI	 10.10.10.134	12	3	0	1mo, 11d ago	4d, 1h ago	
 Madge 5129	Low	aCt5GNbejN8PEtuEKQT4	 10.10.10.134	17	1	13	6d, 20h ago	6d, 20h ago	
 Queen 1447	Medium	Lz46ocalWlHfbKIHjCNw	 10.10.10.149	7	3	0	7d, 3h ago	4d, 2h ago	
 Tamil 2366	Medium	6IAyASDt9J9sVvzMRcz	 10.10.10.134	7	7	0	4d, 2h ago	4d, 2h ago	
 Taupe 2634	Low	JL2kQwXSTODmV1mBfzmg	 10.10.10.218	9	1	0	18d, 21m ago	3h, 29m ago	
 Thomas 4024	Medium	yUAJgoPaS4PJ7W3tyG	 10.10.10.149	3	3	0	4d, 2h ago	4d, 2h ago	

On the far left side of the Spotlight Attackers table is a small icon representing the local threat of the attacker, as it pertains to your site. This is a fast way to scan through the spotlight profiles and determine which ones might pose an immediate threat to your system. The severities range from 0 or None to 4 or High.



NOTE: Throughout the Web UI, you may start to see Spotlight Profiles, indicated by the Spotlight icon next to their Profile name. You can choose to display either Local or Global (Spotlight) names (or both) through the User Preferences screen.

Figure 84: User Preferences: Select Spotlight Name Preference

User Preferences

Skin * Dark (Default) ▾
To accommodate aesthetic preferences, you may select an alternate skin.

Language * English (US) ▾
Where available, we will use localized text throughout the system.

Timezone * UTC ▾
By default, items in the User Interface are shown in UTC. To change this, set this value to your preferred time zone.

Prompt Level * Standard ▾
By default, help text is shown inline. If you are familiar with the product, you may wish to have an abbreviated version.

Spotlight Name Preference * ▾
✓ Display Global Names if Available
Display Local Names Only
Display Both Names, Side by Side (widescreen only)

Auto Refresh ☒
Enable auto refreshing of datasets displayed on pages. (Requires JavaScript)

Refresh Interval * 120
How often, in seconds, automatic refreshes will occur, if 'auto refresh' is enabled.

Records Per Page * 15
The default number of records returned per page. Used for all lists of data throughout the entire UI.

Debug Mode ☐
Certain items in the user interface can be usable for troubleshooting, but are hidden by default. To see them, select this option.

Save

CHAPTER 26

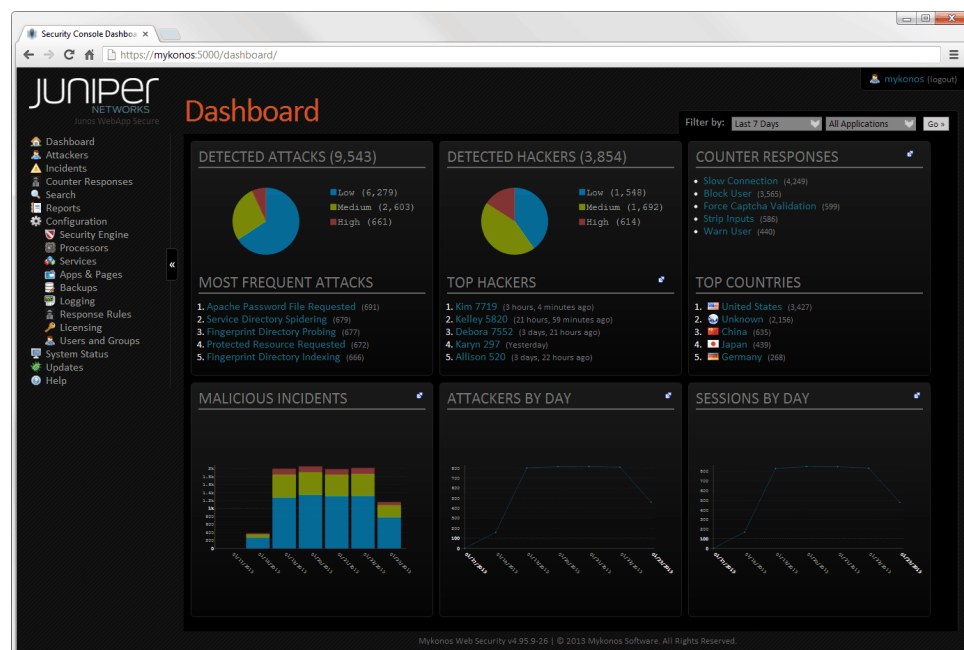
Security Monitor

- [The Dashboard on page 123](#)
- [Attackers on page 128](#)
- [Responses Tab on page 131](#)
- [Locations Tab on page 132](#)
- [Incidents on page 133](#)
- [Counter Responses on page 135](#)
- [Sessions on page 136](#)
- [Search on page 137](#)
- [Reporting on page 139](#)
- [Configuration on page 139](#)
- [System Status on page 140](#)
- [Updates on page 140](#)

The Dashboard

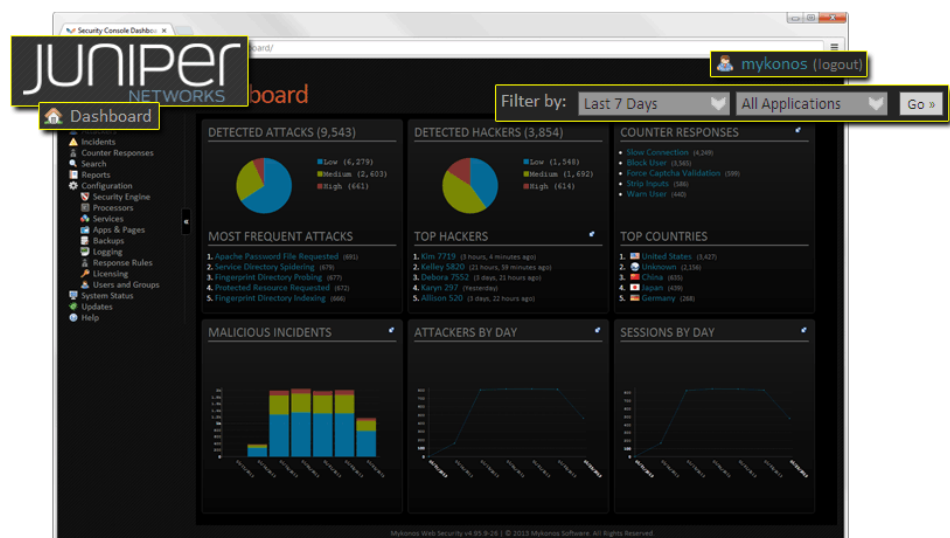
The Dashboard is the page used to display currently monitored incidents, sessions, and responses. It can be reached by navigating to **Dashboard** in the left-hand menu, or simply by clicking the logo in the top left of the window.

Figure 85: Security Monitor Dashboard



The dashboard contains graphs and charts depicting the activity on protected web applications. By default, the dashboard will display the information gathered from all configured applications in the past week (7 days), but you can focus on specific applications or change the date range by using the **Filter By:** tab near the top right of the window.

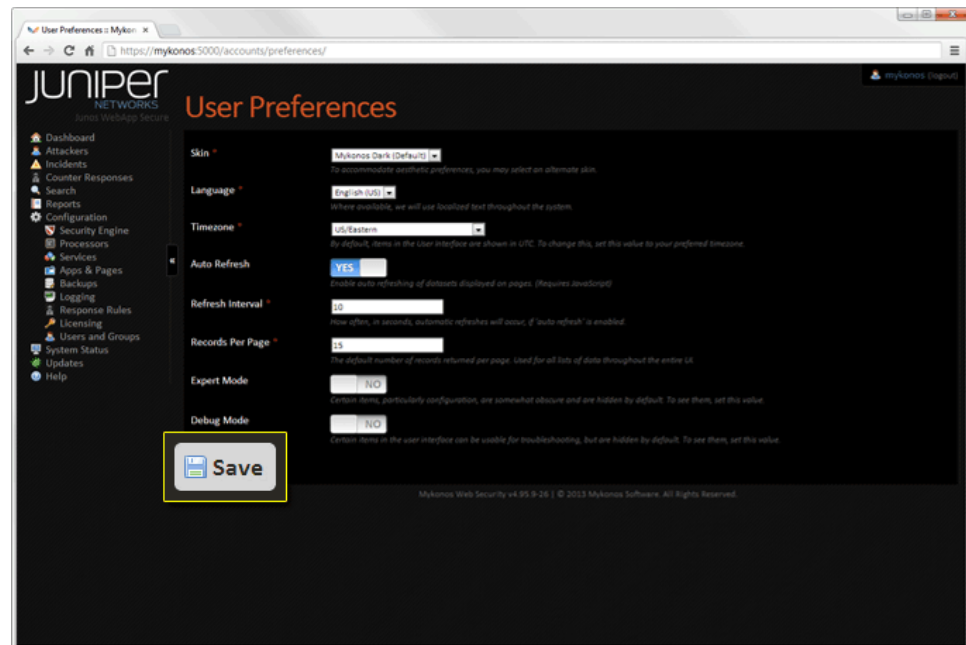
Figure 86: Dashboard - Filter By tab



There are other customization options that can be configured per account by clicking your username in the very top right of the window. Options include setting your timezone, enabling auto-refresh of data contained in the dashboard, and configuring the number

of records returned per page. Change these preferences to your liking and click **Save** at the bottom of the page.

Figure 87: User Preferences



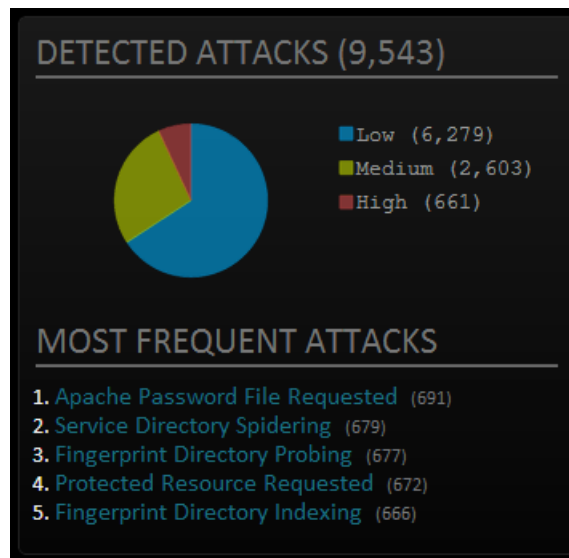
The main portion of the dashboard consists of various panes, each containing information gathered by the Security Engine.



NOTE: The Web UI should be compliant with current browsers. While older browsers might work, we recommend updating to the latest versions for best functionality. Likewise, it is recommended JavaScript be enabled in the browser. JavaScript is used in the UI to enhance functionality and usability, and while browsing without JavaScript is possible, it is not recommended. The UI targets screen resolutions of 1366x768 or higher for normal operations, and targets 1440x900 when debug mode is enabled.

Table 8: Security Monitor Dashboard Panes

Detected Attacks This pane displays a chart that contains all incidents created (within the filter parameters) segregated by complexity of the attacks. It is a good way to visualize how active your web site's attackers are. You can hover over each portion of the pie chart to display the actual number of attacks for each complexity. This pane also contains a short list of the most frequent attacks. Clicking on any attack in the list will open the Incident Type page for that particular incident.



Detected Hackers The Detected Hackers pane displays information on actual hacker profiles created within the specified time frame. Each hacker gets a skill level which segments the pie chart. As in any other chart, you can hover over the pie chart to view specific counts. Additionally, the most active hackers are displayed in a list below the chart. Clicking on any of these hackers will open the Hacker Profile page for that particular profile.

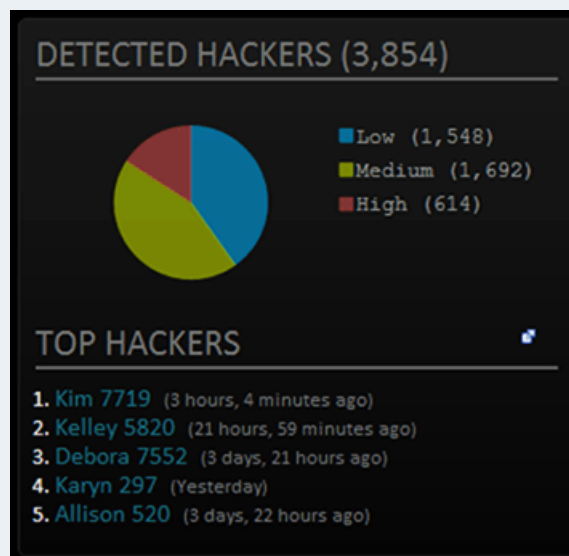


Table 8: Security Monitor Dashboard Panes (*continued*)

Counter Responses The Counter Responses pane lists the top responses that have been recently triggered by WebApp Secure. The lower portion of the pane lists countries in descending order by response count. You can click on any of the counter responses to open the Counter Response Type page for that response (explained later), or click on the specific country to find other information WebApp Secure has gathered on that country.



Malicious Incidents The Malicious Incidents dashboard pane consists of a chart depicting the number of incidents over time. It also stacks these incidents by complexity.

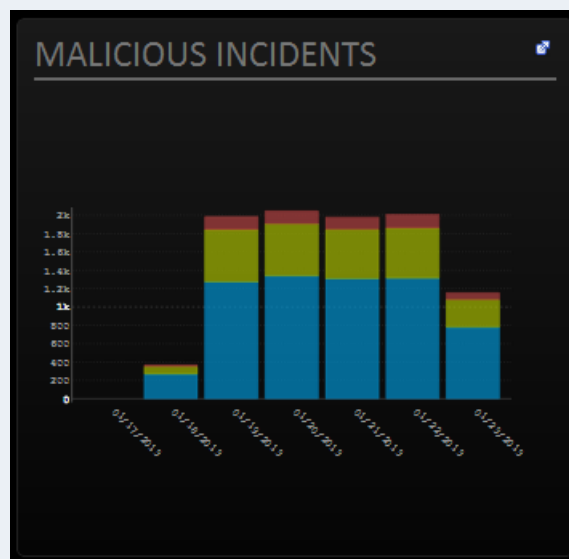
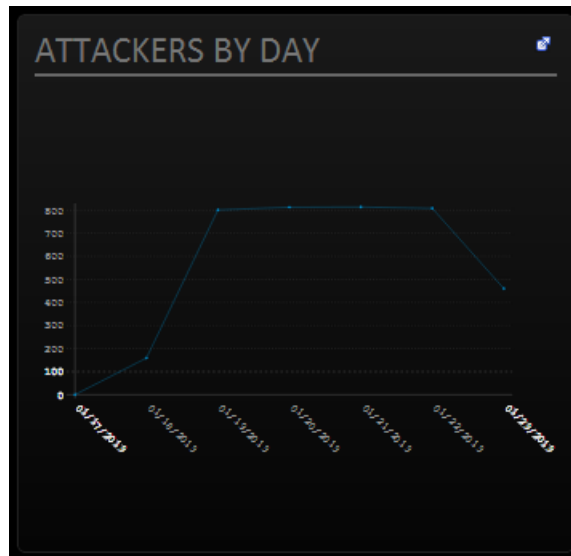
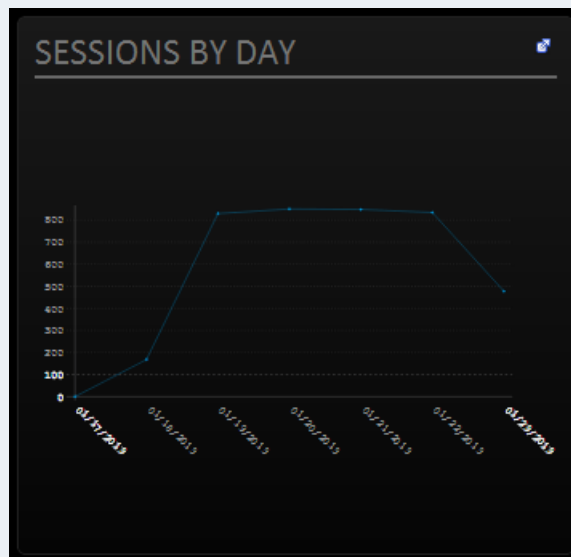


Table 8: Security Monitor Dashboard Panes (*continued*)**Attackers By Day (Attackers By Hour)**

This pane contains a graphical representation of how many detected hackers were active on the protected site, separated by day (or separated by hour if the filter "Last Day" is currently set).

**Sessions By Day (Sessions By Hour)**

Similar to the Attackers By Day pane, the Sessions By Day pane shows the number of sessions active on the protected site each day (or each hour if the filter "Last Day" is set).

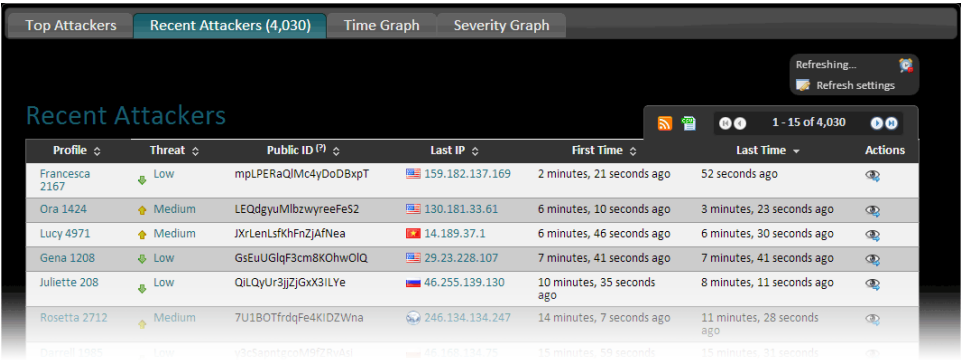


NOTE: If you would like more horizontal space on any page, you can collapse the navigation menu by clicking on the double arrow (<<) button to the right of the menu.

Attackers

The Attackers page contains any information on profiled attackers, and can be accessed by clicking **Attackers** in the left navigation menu.

Figure 88: Recent Attackers



Profile	Threat	Public ID	Last IP	First Time	Last Time	Actions
Francesca 2167	Low	mplPERaQIMc4yDoDBxpT	159.182.137.169	2 minutes, 21 seconds ago	52 seconds ago	
Ora 1424	Medium	LEQdgyuMlbzwyreeFeS2	130.181.33.61	6 minutes, 10 seconds ago	3 minutes, 23 seconds ago	
Lucy 4971	Medium	JXrLenLsfKhFnZjAfNea	14.189.37.1	6 minutes, 46 seconds ago	6 minutes, 30 seconds ago	
Gena 1208	Low	GsEuUGlqF3cm8KOhwOIQ	29.23.228.107	7 minutes, 41 seconds ago	7 minutes, 41 seconds ago	
Juliette 208	Low	QilQyUr3jZjGxX3ILYe	46.255.139.130	10 minutes, 35 seconds ago	8 minutes, 11 seconds ago	
Rosetta 2712	Medium	7U18OTfrdqFe4KIDZWna	246.134.134.247	14 minutes, 7 seconds ago	11 minutes, 28 seconds ago	
Daniel 1885	Low	y3cFapetpnaMR7PuKd	46.108.134.75	15 minutes, 50 seconds	15 minutes, 51 seconds	

There are various data views you can navigate through via the tabs near the top of the page. You can also search for attackers by using the search field in the upper right side of the page, under the **Filter** widget.

- **Top Attackers** This tab contains an ordered list of the most active attackers, calculated based on a weighting algorithm that takes into account the number of incidents and their corresponding complexities.
- **Recent Attackers** This tab displays a table of the most recent profiles active on the protected system. Each row consists of the Profile name, Threat level, their Public ID (for use with the Support Processor), the Last IP they used on the system, the First Time and Last Time they attacked the system, and available actions for that profile. Clicking on the "eye" icon or the profile name will lead you to the page for that particular profile. You can also click on a threat level to view other attackers with similar threat, and you can click on a Last IP to navigate to the Location page for that IP. To keep this data fresh, the monitor will periodically refresh the page (if Auto-refresh is enabled in the User Preferences). To stop this from happening, click the alarm clock icon in the top right corner of the tab to stop refresh.
- **Time Graph** The Time Graph is a larger version of the same line graph displayed on the Dashboard.
- **Severity Graph** This graph is a larger version of the same pie graph displayed on the Dashboard.



NOTE: At any point on this page, you can click on an attacker's given name to navigate to that Attacker's Profile page.

Figure 89: Attacker Profile



The Attacker Profile page displays any information that pertains to a particular attacker. At the top of the page you will see the Attacker Card, which contains a short overview of the profile. This card contains the attacker's assigned name, last IP used, the first and last date the attacker was active, and the Public ID of the attacker, for use with the Support Processor in unblocking that profile. On the right side of the card there is a threat gauge that indicates the current threat of that attacker, where green, yellow, and red indicate low, medium, and high threat, respectively. The severity icons are displayed as follows:

- (n/a): 0.0 - None
- : 1.0 - Suspicious
- : 2.0 - Low
- : 3.0 - Medium
- : 4.0 - High

Available on the right side of the Attacker Profile page is a quick **Actions** box, where you can rapidly perform various profile-related functions such as blocking the attacker, warning the user, editing the profile, and deleting the profile.



NOTE: Deleting the profile will essentially erase all information gathered on that attacker, and will effectively remove all blocks or other responses on that profile.

Underneath the attacker card and quick actions box is a series of tabs, where all of the attacker's specific activity information resides. The Incidents tab contains a list of all incidents triggered by that attacker. The Incident name, complexity, count, first and last time triggered are all available for each item in the list. Additionally you can click the Details icon (the eye) to view more information about any particular incident.

Responses Tab

The Responses tab contains information relating to all of the active and inactive responses issued to that attacker. Each entry contains the actual name of the response issued, the configuration (if any) used when issuing the response, the time the response was created, the delay set (if any), the time the response expires (if at all), the time the response was finally deactivated (if it has been deactivated).

If the response is active, you can click the Deactivate Response icon (the stop sign under Actions) to deactivate the response instantly. Alternatively, you can click the **Deactivate Selected** button or to deactivate all responses, click the **Deactivate All** button.

Figure 90: Responses tab - Deactivate

Response	Config	Created	Delayed	Expires	Deactivated	Actions
<input type="checkbox"/> Google Map		Just now			Never	
<input type="checkbox"/> Slow Connection	max=6000, min=2500	4d, 22h ago		3d, 22h ago	3d, 22h ago	
<input type="checkbox"/> Force Captcha Validation		4d, 22h ago			4d, 22h ago	

Deactivate Selected Deactivate All

It is in this tab that you can manually activate Counter Responses on the current attacker. The available counter responses are:

- **Block User** To block the user from accessing the protected application completely, you can activate the Block User counter response. The next time the attacker tries to visit any page on the application, they will see a configurable message indicating they have been blocked from accessing the content. If the Support Processor is enabled, they are also given their Public ID (also shown on the Attacker Profile page for that profile) that they can give to support if they feel the block was in error.
- **Filter on SRX** For more information on what this counter response does, see: SRX Integration. In jest, it feeds a message to an SRX device that can handle traffic at the network level.



NOTE: This counter response can be activated without configuring an external network device, but it will not do anything. WebApp Secure requires a properly configured external device for this counter response to function properly.

- **Break Authentication** Hashes any incoming passwords when attempting to login, effectively thwarting brute-force attacks that have correct credentials. Even with the correct password, the login will be unsuccessful.
- **Cloppy** Activating this counter response will activate an animated paper clip that intimidates the user with configurable messages. For information on how to customize this response, see the Cloppy Processor in Processor Reference section.
- **Force Captcha Validation** The user will be prompted with a Captcha that has to be solved to continue using the web site.
- **Google Map** The user will be shown a map of lawyers near their determined location. The search term fed into Google Maps can be configured, see the Google Map Processor in the Processor Reference section.
- **Inject Header** The suspected hackers requests will have a custom header injected into them, useful for tracking.
- **Logout User** Terminates any current user sessions for this profile on a site.
- **Slow Connection** The user's requests to the site will be delayed by a configurable window of milliseconds. This can frustrate the attacker and cause them to abandon their future attacks. This response can take a `<config/>` node with 'min' and 'max' parameters, for example; `<config min=1000 max=5000 />` will slow the attackers requests by 1 to 5 seconds.
- **Strip Inputs** If you suspect the attacker's inputs shouldn't be trusted (such as those inputs submitted in forms on the site), you can choose to activate this response which will strip them from all incoming requests. This will also strip any query parameters from the request url as well.
- **Warn User** The next request sent by the attacker will respond with a pop-up warning message that lets the attacker know he/she is being watched. The warning message can be configured, see the Warning Processor in the Processor Reference section.

Consecutive requests might be grouped together and are viewable via the Sessions tab. Each entry in this tab contains the Remote Address used during the session (the IP), the Browser and Operating System used during the session, the number of Requests made and Pages returned during that session, the number of Errors generated by the server in response to requests in that session, as well as the First and Last Active times. You can also click on the **Details** icon (the eye) to view more information about any particular session.

Locations Tab

The Locations tab contains a list of all locations used by the attacker. For each location, you are able to see the Remote Address (IP) associated with that location, the City,

Region, and Country associated with the location (if they can be found), and the First and Last Active times for the location. Depending on the location, you might also be able to load a map showing that location (if it can be determined) by clicking on the **Map** icon. You can also click on the **Details** icon (the eye) to view more information on any particular location, including all other attackers that were found to be using the same location, and other Incidents, Sessions, or Environments used in conjunction with that location. If WebApp Secure can determine the attacker was using a specific Browser and Operating System combination, an entry in the Environments tab will be added. Each entry contains the Browser and Operating System used, along with the full User Agent string and First and Last active dates. If you wish to find other attackers that used the same Environment, click on the magnifying glass icon. This will bring you to a page where you can see other Attackers that used this Environment, Incidents produced with this Environment, Sessions found that were using this Environment, and Locations that used this Environment.

Incidents

The Incidents page contains any information on specific incidents that have been triggered, and offers additional information on all of the incidents that can be detected by WebApp Secure.

Incident	Attacker	Complexity	Count	First Time	Last Time	Actions
Service Directory Spidering	Karl 3462	Medium	1	1 minute, 35 seconds ago	1 minute, 35 seconds ago	[Details] [Map]
Apache Password File Requested	Karl 3462	Low	1	2 minutes, 28 seconds ago	2 minutes, 28 seconds ago	[Details] [Map]
Captcha Directory Indexing	Lynnette 1107	Low	1	3 minutes, 28 seconds ago	3 minutes, 28 seconds ago	[Details] [Map]
Apache Password File Requested	Lynnette 1107	Low	1	4 minutes, 31 seconds ago	4 minutes, 31 seconds ago	[Details] [Map]
Service Directory Indexing	Pam 4739	Medium	1	5 minutes, 29 seconds ago	5 minutes, 29 seconds ago	[Details] [Map]

There are various data views you can navigate through via the tabs near the top of the page. You can also search within Incidents by using the search field in the upper right side of the page, under the **Filter** widget.

- **Most Common** This tab displays a list of the most frequently triggered incidents in descending order. Count of triggered incidents of that type is displayed to the right of each item in the list, and a graphic depicting the complexity of that incident is visible to the left. Clicking on a particular incident in this list will bring you to a page with additional information on that incident. By default, WebApp Secure only displays malicious incidents (those that might be of direct interest to WebApp Secure users). If you wish to show all incidents triggered, you can click on the **Show all incidents** link above the list.
- **Most Recent** This tab displays a table of the most recent incidents triggered. The incident name is displayed along with the profile that triggered the incident, the complexity of that incident, the Count indicating the number of times that incident was triggered at one time (using the same data), the first and last times the profile activated that particular incident, and any actions available to the WebApp Secure user regarding that incident. You can navigate to other pages by using the tab above the table. Here you can jump to the next page, previous page, first page, and last page by using the corresponding buttons. You can also jump to a specific page or change the number of rows returned per page by clicking on the label between the navigation buttons. By default, only malicious incidents are displayed. To display all malicious and non-malicious incidents, click the **Show all incidents** link above the title. To keep this data fresh, the monitor will periodically refresh the page (if Auto-refresh is enabled in the User Preferences). To stop this from happening, click the alarm clock icon in the top right corner of the tab to stop refresh.
- **Browse by Complexity** For informational purposes, this tab allows you to browse the list of detectable incidents, grouped by complexity. Clicking on an incident will bring you to an informational page that contains a description of that incident, and allows you to search for triggered incidents of that type.
- **Time Graph** The Time Graph is a larger version of the same bar graph displayed on the Dashboard.
- **Severity Graph** This graph is a larger version of the same pie graph displayed on the Dashboard.



NOTE: Clicking on a particular incident's name will bring you to the Incident Details page for that incident. On this page all information about that particular incident is shown.

Incident Details

Incidents » Incident Detail

Apache Configuration Requested

Attacker: Lesa 9502
Location: Columbus Ohio, United States
Environment: Internet Explorer 9.0, Windows
Session: 130.90.162.56
Last Time: 16 days, 19 hours ago **First Time:** 16 days, 19 hours ago

Description Details Request Response

CAUSE:

Apache is a very common web server. As a result, hackers will often look for vulnerabilities specific to Apache, since there is a good chance that any given website is running Apache. One such vulnerability involves the use of an .htaccess file to provide directory-level configuration (password-protected resources, directory indexing options, etc...), while not sufficiently protecting the .htaccess file itself. By convention, configuration files should not be exposed to the public — so if a user requests .htaccess or a related resource, they should get either a "404 Not Found" or "403 Forbidden" error. Unfortunately, an improperly-configured installation of Apache may not block requests for these resources. In such a scenario, a hacker could gain valuable knowledge of the way the server is configured.

Junos WebApp Secure will automatically block any requests for the .htaccess resource, and instead return a fake version of the file, which contains the directives necessary to password-protect a fake resource. It is safe to assume the request is malicious because no legitimate user should ever be requesting this resource.

Threat Level: Low

Near the top of the page there is an incident infobox that contains a summary of the incident, including the Attacker that caused the incident, the Location and Environment that attacker was using, the Session (IP) used when triggering the incident, and the First and Last times that particular incident occurred. Underneath the infobox there is a series of tabs that display the Description of the Incident type, Details for the incident (differs from incident to incident), and the raw Request and Response objects.

Counter Responses

The Counter Responses page contains any information on the various responses WebApp Secure can issue to potential threats. It contains the following tabs:

Browse by Type **Active Responses (5,492)** Inactive Responses (4,439)

Refresh in... 2 Refresh settings

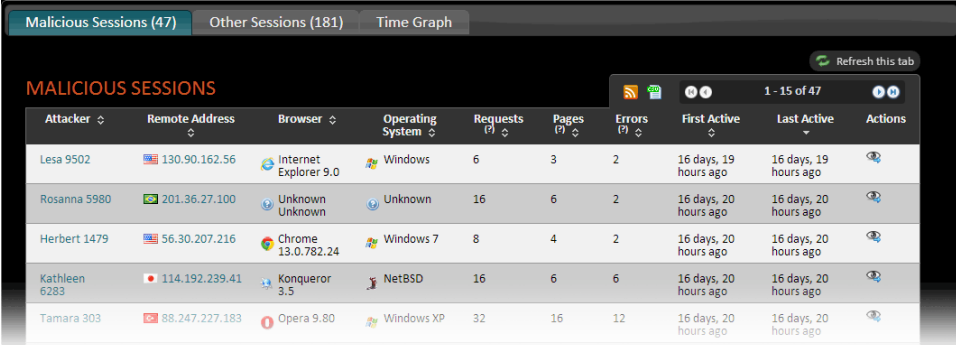
ACTIVE RESPONSES

Attacker	Response	Config	Created	Delayed	Expires	Actions
Tammie 6840	Block User		44 seconds ago		In 4 days, 23 hours	
Karl 3462	Block User		2 minutes, 48 seconds ago		In 4 days, 23 hours	
Karl 3462	Slow Connection	max=6000, min=2500	3 minutes, 47 seconds ago		In 23 hours, 56 minutes	
Lynnette 1107	Block User		4 minutes, 40 seconds ago		In 23 hours, 55 minutes	
Lynnette 1107	Slow Connection	max=6000, min=2500	4 minutes, 40 seconds ago		In 23 hours, 55 minutes	
Lynnette 1107	Slow Connection	max=6000, min=2500	5 minutes, 48 seconds ago		In 23 hours, 54 minutes	
Mara 8293	Slow Connection	max=6000, min=2500	6 minutes, 27 seconds ago		In 23 hours, 53 minutes	
Paul 4758	Block User		6 minutes, 47 seconds ago		In 4 days, 23 hours	

- **Browse by Type** In this tab, you can view information on any of the counter responses WebApp Secure can issue. Clicking on a specific response will take you to a page that explains that response, and allows you to search for profiles that were issued that response.
- **Active Responses** In the Active Responses tab, a table displays the most recently activated responses, along with the profile that the response was issued on, the specific response issued, any configuration used in that response (blank if there wasn't any), the time the response was issued, how long to delay the response.
- **Inactive Responses** The Inactive Responses tab is formatted like the Active Responses tab, but shows all responses which have been deactivated, either manually or due to response expiration.

Sessions

When users browse the protected site, similar or back-to-back requests can be grouped together in a Session. The Sessions page allows you to view each of these browsing sessions.



Attacker	Remote Address	Browser	Operating System	Requests	Pages	Errors	First Active	Last Active	Actions
Lesa 9502	130.90.162.56	Internet Explorer 9.0	Windows	6	3	2	16 days, 19 hours ago	16 days, 19 hours ago	
Rosanna 5980	201.36.27.100	Unknown Unknown	Unknown	16	6	2	16 days, 20 hours ago	16 days, 20 hours ago	
Herbert 1479	56.30.207.216	Chrome 13.0.782.24	Windows 7	8	4	2	16 days, 20 hours ago	16 days, 20 hours ago	
Kathleen 6283	114.192.239.41	Konqueror 3.5	NetBSD	16	6	6	16 days, 20 hours ago	16 days, 20 hours ago	
Tamara 303	88.247.227.183	Opera 9.80	Windows XP	32	16	12	16 days, 20 hours ago	16 days, 20 hours ago	

The tabs available in the Sessions page show Malicious Sessions, Other (non-malicious) sessions, and a graph of sessions over time. Each Session entry contains information including the Attacker the session belongs to (if it was a session with malicious intent), the Remote Address used during the session (the IP), the Browser and Operating System used during the session, the number of Requests made and Pages returned during that session, the number of Errors generated by the server in response to requests in that session, as well as the First and Last Active times. You can also click on the Details icon (the eye) to view more information about any particular session.

Clicking on the **Details** icon will bring you to the **Session Details** page for that session. On this page all information about that particular session is shown.

Session Detail

Sessions » Session Details

Attacker: Vera 105
Last Remote Address: 10.38.61.25
Last Location: Local Network
Last Environment: Opera 9.80 Linux

Requests: 9 **Pages:** 2 **Errors:** 2
Last Active: 16 days, 20 hours ago **First Active:** 16 days, 20 hours ago

Incidents (1) Locations (1) Environments (2)

INCIDENTS
 Showing malicious incidents only. Show all incidents

Incident	Attacker	Complexity	Count	First Time	Last Time	Actions
Apache Configuration Requested	Vera 105	Low	1	16 days, 20 hours ago	16 days, 20 hours ago	

Near the top of the page there is a session infobox which contains a summary of the session, including the Attacker associated with the session, the Last known address (IP) used in conjunction with the session, the Last Location and Environment used during the session, and information regarding the number of Requests issued, Pages returned, and Errors generated by the server as a result of a request. Underneath the infobox is a series of tabs that display other Incidents, Locations, and Environments used during this browsing session.

Search

You can find a particular attacker, incident, or session by using the search functionality in the security monitor. To search, type the keyword in the Query form field, and optionally modify the desired date-range (last 7 days by default), applications (all applications by default), and the scope of your search. The scope can include Attackers, Incidents and/or

Sessions. Depending on the complexity of your search parameters, it might take a couple seconds to complete. Once finished, the results will be displayed.

Search

Query *

What do you want to search for?

Date Range *

How far back to you want to search?

Application *

Which application do you want to filter by?

Scope *

Where do you want to search for it?


Max Results/Scope *

The maximum number of records to return, per scope.


 **Search**

Search Results (3)


ATTACKERS (1)

Profile Name	Threat Level	Public ID (?)	Last IP	First Time	Last Time	Actions
Ella 2360	Low	HC3YION6RztfdlIY2Dky	10.10.10.113	2 days, 21 hours ago	2 days, 21 hours ago	

INCIDENTS (1)

Incident	Attacker	Complexity	Count	First Time	Last Time	Actions
Apache Configuration Requested	Ella 2360	Low	2	2 days, 21 hours ago	2 days, 21 hours ago	

SESSIONS (1)

Attacker	Remote Address	Browser	Operating System	Location	Requests	Errors	First Active	Last Active	Actions
Ella 2360	10.10.10.113	Chrome 26.0.1410.43	Windows 7	Local Network	2	0	2 days, 21 hours ago	2 days, 21 hours ago	

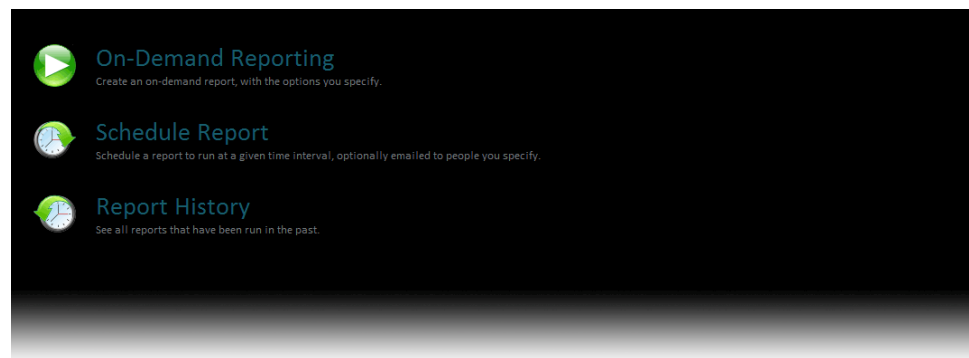
The following items are indexed in the search (meaning if the string matches any items in these categories, it is displayed.):

- User Agent
- Browser Name
- Browser Version
- Incident Name
- IP Address
- Host
- Geographic Region

- Geographic City
- Geographic ZIP
- Country Name
- Country Code
- Profile Name
- Profile Description
- Profile Public Key
- Incident Request Content
- Incident Response Content

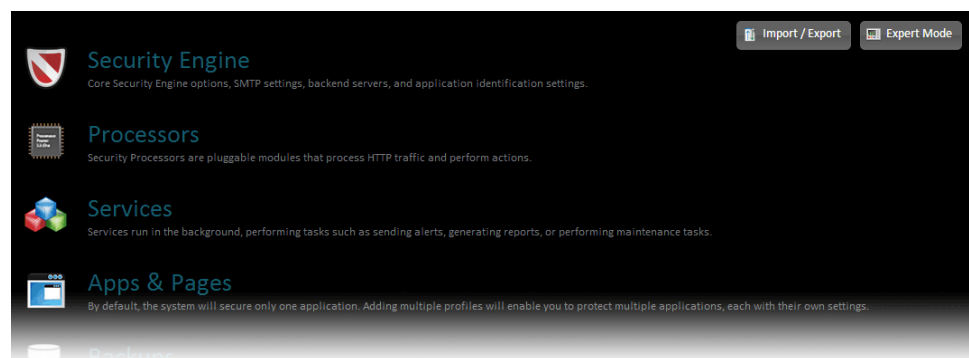
Reporting

The Reports page is responsible for producing graphical and textual representations of the activity passed through WebApp Secure.



Configuration

Configuration section of the security monitor allows you to change numerous aspects of the software.



System Status

System Status allow you to view performance metrics of your installation. This includes information on system health, running services, and the routing table.

Figure 91: System Status

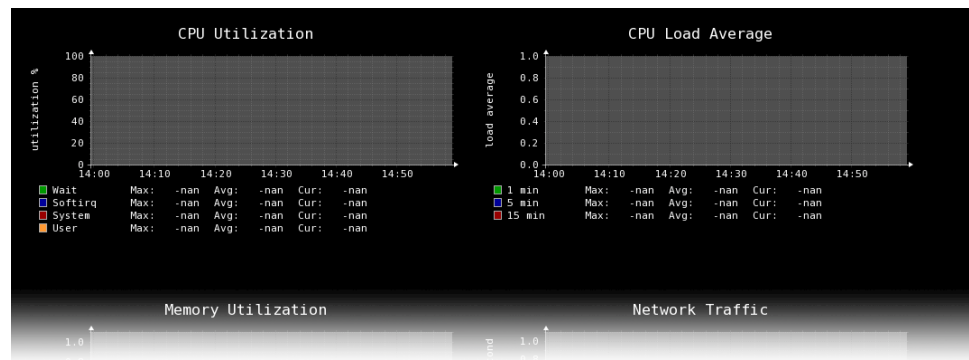


Figure 92: Services Status

RRD: localhost Backend Health Services Routing Table				
Name	Description	Status	PID	Actions
mykonos-api	Mykonos API	✓ Running	3940	↺ ↻ ↻
mykonos-cluster-services	Mykonos Cluster Services	✓ Running	4043	↺ ↻ ↻
mykonos-datastore	Mykonos Datastore	✓ Running	3877	↺ ↻ ↻
mykonos-pyro	Mykonos Pyro	✓ Running	4272	↺ ↻ ↻
mykonos-reports-api	Mykonos Reports API	✓ Running	4080	↺ ↻ ↻
mykonos-security-engine	Mykonos Security Engine	✓ Running	4136	↺ ↻ ↻
mykonos-services	Mykonos Local Services	✓ Running	3983	↺ ↻ ↻
mykonos-ui	Mykonos Web UI	✓ Running	3973	↺ ↻ ↻
nginx	Routing Proxy	✓ Running	5292	↺ ↻ ↻
nginx_management	Management Proxy	✓ Running	3866	↺ ↻ ↻

Updates

Perform updates to the installation by navigating to this page.

Figure 93: Updates

ONLINE UPDATES
Automatic updates download is enabled. Disable Manually Check for Updates
There are no updates ready for installation at this time.

OFFLINE UPDATES
If you would like to apply an update manually, you may use this form to upload the file you have received.

File * No file chosen

Start after upload? ☐
After uploading is complete, whether or not to start the update process.

Upload File

CHAPTER 27
















Autoresponse Defaults and Rule Creation

- Autoresponse Overview on page 143
- Editor Overview on page 147

Autoresponse Overview

An autoresponse is composed of a set of rules which define the conditions under which a counter response should be automatically created and activated for a specific session or profile. It is possible to have as many rules as needed to protect the system. However, the more rules, the longer it will take to determine if a new incident matches an event condition. In addition, the more conditions in the rule, the longer the rule will take to evaluate if the event condition matches a new incident.

Figure 94: Autoresponse

Name	Status	Description	Actions
Session Management	 	This autoresponse rule handles the core session management incidents generated by the security engine. read more	
Processor: ETag Beacon	 	This autoresponse rule handles incidents generated by the etag beacon processor. These incidents are generally triggered when a user attempts to exploit the tracking mechanism used by the application to re-identify users. This rule is designed to first slow the user's connection down, and, if the behavior continues, slow it down even further. read more	
Processor: Application Vulnerability Processor	 	This autoresponse rule handles incidents generated by the application vulnerability processor. These incidents are generally triggered when a user attempts to exploit a known vulnerability in a 3rd party application. This rule is designed to first slow the user's connection down, and, if the behavior continues, break the application with the clear input response. read more	
Processor: Access Policy	 	This autoresponse rule handles incidents related to the access policy processor. These incidents are generally triggered by malicious spiders and the rule is designed to block those spiders for a prolonged period of time. read more	
Processor: Basic Authentication	 	This autoresponse rule handles the various incidents triggered by the basic authentication processor. These incidents are triggered by users	



NOTE: You can view the default responses for each rule by clicking [read more](#) at the bottom of the entry's description in the UI.

Table 9: Autoresponse Descriptions

Default Autoresponse	Description
Session Management	This autoresponse rule triggers if the user attempts to manipulate the WebApp Secure session tracking cookie.
Application Vulnerability Processor	If your web-application uses supported 3rd party applications (like Joomla, Wordpress, etc.), this processor will analyze and act on malicious traffic that intends to exploit them. For more information on which 3rd party tools are supported, refer to the AutoResponse documentation in Security Monitor.
Login Processor	This rule triggers on incidents that are generally triggered by abusive and suspicious activity targeted at the web sites authentication system.
Access Policy Processor	This autoresponse rule triggers if the user attempts to exploit the fake service exposed by this processor.
ETag Beacon Processor	This autoresponse rule triggers if a user attempts to manipulate the WebApp Secure cached based tracking token.
Basic Authentication Processor	This autoresponse rule triggers when the user attempts to exploit the fake .htaccess file exposed by this processor.
Robots Processor	This autoresponse triggers when the user or malicious spider uses the information in the robots.txt file for illegitimate purposes.
Hidden Input Form Processor	This autoresponse rule triggers when the user modifies a hidden form input parameter.
Cookie Processor	This autoresponse rule triggers when the user attempts to manipulate the value of a cookie.
AJAX Processor	This autoresponse rule triggers when the user interacts with a fake AJAX function injected into the web application. If the user reverse engineers the code and manually invokes its behavior, such as would happen with an automated script or spider, the rule will fire. If the user actually invokes the Javascript function, the rule will fire.
Header Processor	This autoresponse rule triggers when the user has unusual headers or header data which a normal browser or well developed spider would not supply. If the user excludes required headers such as Host and UserAgent, manipulates their user agent header, overflows headers beyond RFC standards will cause this rule to activate.
Hidden Link Processor	This autoresponse rule triggers when a spider or malicious user attempts to identify unreferenced resources in a fake directory.
Query Parameter Processor	This autoresponse rule triggers when a user manipulates the fake query parameter injected by the system more than 3 times.
Method Processor	This autoresponse rule triggers when a user or spider sends a request with a malicious HTTP method such as TRACE.
Error Processor	This autoresponse rule triggers when a user attempts to find unreferenced resources by guessing file names.

Table 9: Autoresponse Descriptions (*continued*)

Default Autoresponse	Description
File Processor	This autoresponse rule triggers when a user attempts to find sensitive files by guessing file names or changing parts of valid file names.
Warning Processor	This autoresponse rule triggers when a user attempts to automate the dismissal of the warning response.
Cookie Protection Processor	This autoresponse rule triggers when a user attempts to modify the web application session cookie.
Captcha Processor	This autoresponse rule triggers when a user attempts to find a way to bypass the captcha response without solving the captcha.
CSRF Processor	This autoresponse rule triggers if a user attempts to manipulate the CSRF protection introduced by the system, potentially to find a filter evasion vulnerability.
Custom Authentication Processor	This autoresponse rule triggers if a user attempts to exploit the authentication mechanism offered by the system.
Client Beacon Processor	This autoresponse rule triggers when the user attempts to tamper with the client side tracking logic.
New and Modified Profiles	This autoresponse rule sends out an alert any time a new profile is created, or a profile elevates its threat level. The severity of the alert will equal the threat of the new or elevated profile that triggered the alert.
Returning Profile	This autoresponse rule sends out an alert any time a profile returns on a subsequent day. For example, a new hacker is observed on Monday, if the hacker is only active for 1 hour on Monday, but returns on Tuesday to continue, this rule will issue an alert. The severity of the alert will equal the threat level of the profile.
New Incident	This autoresponse rule sends out an alert any time a new incident is observed. The severity of the alert will equal the complexity of the incident.
New Response	This autoresponse rule sends out an alert any time a new counter response is activated. The severity of the alert will always equal 1.

Editor Overview

To create an autoresponse, open the configuration UI and select the **ADD New Rule** button. This will launch the editor which can be used to create and edit an autoresponse.

Table 10: Autoresponse Editor Fields

Field	Description
Name	The name of the autoresponse.
Description	Description of the autoresponse and its triggers.
Enabled	Sets an autoresponse to be active.
Safe Mode	Allows the autoresponse to activate, but does not actually respond. This setting is for testing and debugging autoresponses.
Code	The actual code that defines the autoresponse.

Table 10: Autoresponse Editor Fields (*continued*)

Field	Description
Events	The events that will trigger the autoresponse.
Log	A table which consists of any log statements printed during autoresponse execution. Use the JavaScript console object to output to an autoresponse's log.
API Reference	A link to the Autoresponse API documentation.

PART 4

Monitoring

- [The Processors on page 151](#)
- [Honeypot Processors on page 155](#)
- [Activity Processors on page 181](#)
- [Tracking Processors on page 207](#)
- [Response Processors on page 219](#)
- [Incident Methods on page 267](#)
- [Captcha Template on page 271](#)
- [Log Format on page 275](#)

CHAPTER 28

The Processors

- [Complexity Rating Definitions on page 151](#)
- [Security Engine Incidents on page 152](#)
- [Session Cookie Spoofing on page 152](#)
- [Session Cookie Tampering on page 152](#)
- [Security Processors on page 153](#)

Complexity Rating Definitions

Complexity is a rating of the skill, effort, and experience necessary to trigger a specific incident. The following is a description of the rating system:

- **Informational (0.0):** Informational incidents represent information about the client that may or may not indicate malicious activity, but are not common. Informational incidents are used to identify more complex abuse patterns that cannot be identified from a single request. An example of an informational incident is when the user has disabled the Referer header.
- **Suspicious (1.0):** Suspicious incidents represent activity that is abnormal but not guaranteed to be malicious. This is similar to an informational incident, except that the event is borderline malicious, not just unusual. Just like informational incidents, suspicious incidents are used to identify more complex abuse patterns that cannot be confirmed as malicious from just one request. An example of a suspicious incident is when the user requests a file that does not exist (404 error).
- **Low (2.0):** Low complexity incidents represent malicious activity that does not require any special tools, does not require a deep understanding of application architecture, and generally can be executed by an unsophisticated threat. An example of a low complexity incident is when the user modifies a query string parameter in the URL.
- **Medium (3.0):** Medium complexity incidents represent malicious activity that would require special tools, advanced browser configuration, scripting, or a understanding of how web applications are designed and implemented. These types of attacks are generally not executed by unsophisticated attackers, and are more likely to be targeted at the protected site, rather than at an arbitrary IP range. An example of a medium

complexity incident is when the user requests the robots.txt spider configuration file from a browser or a script spoofing its identity as a browser.

- **High (4.0):** High complexity incidents represent malicious activity that is highly advanced and requires a deep understanding of web application architecture, implementation, security features, and multi request workflows. High complexity incidents are generally far too advanced for an average attacker and usually have a specific target. An example of a high complexity incident is when a user is able to break the encryption used on basic authentication password files.

Security Engine Incidents

While the majority of all incidents will be produced by a processor, a few of them are handled by the Security Engine directly. These will be found in the UI under **Session Management** in the **Response Rules** page, and can be enabled or disabled through **Configuration > Security Engine > Incident Monitoring**.

Session Cookie Spoofing

Complexity: Low (2.0)

Default Response: 1x = Logout User, 2x = 1 Day Clear Inputs, 3x = 5 Day Clear Inputs

Cause: WebApp Secure uses an HTTP cookie as one of the components of its fingerprinting technology. The session cookie is comprised of an AES-encrypted and base64-encoded numerical ID and a validation signature. Because the cookie has its own embedded digital signature, any attempt to fabricate or modify a session cookie will almost always result in a corrupted signature. If WebApp Secure detects that a cookie being provided has an invalid signature, but otherwise uses the correct format, it will trigger a "Session Cookie Spoofing" incident.

Behavior: Session cookies are commonly used by a web application order to facilitate state. HTTP, by itself, is not a stateful protocol, and without technologies like cookies, a web application would be unable to correlate requests made by the same user. When an attacker attempts to modify a cookie, especially when they are careful to follow the same format constraints as the original value (22 letters and numbers, or 16 hex characters, etc), they are attempting to modify their state. If for example, an attacker were able to successfully guess the session cookie value of another actively logged in user, they would be able to assume that user's state (including their authentication and authorization levels). This is referred to by the WASC as a "Credential and Session Prediction" attack (see [Credential and Session Prediction](#) for information.)

Session Cookie Tampering

Complexity: Medium (3.0)

Default Response: 1x = Logout User, 2x = 1 Day Clear Inputs, 3x = 5 Day Clear Inputs

Cause: WebApp Secure uses an HTTP cookie as one of the components of its fingerprinting technology. The session cookie is comprised of an AES-encrypted and base64-encoded

numerical ID and a validation signature. Because the cookie has its own embedded digital signature, any attempt to fabricate or modify a session cookie will almost always result in a corrupted signature. If WebApp Secure detects that a cookie being provided does not have a valid signature, and does not follow the correct format, it will trigger a "Session Cookie Tampering" incident.

Behavior: Session cookies are commonly used by a web application order to facilitate state. HTTP, by itself, is not a stateful protocol, and without technologies like cookies, a web application would be unable to correlate requests made by the same user. However, just like form parameters and query string parameters, cookies represent another type of user-input. Just about any attack that can be accomplished by injecting malicious values into a form input (SQL injection², XSS³, Buffer Overflow⁴, Integer Overflow⁵, etc.), could also potentially be accomplished by injecting malicious values into the session cookie. An aggressive hacker would likely test for multiple vulnerability types in all form inputs, query parameters, and cookies, because these are the inputs most likely to be insecurely handled.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Security Processors

The Security Processors are separated into four groups:

- Honeypot Processors
- Activity Processors
- Tracking Processors
- Response Processors

Honeypot processors contain the logic of injecting the fake vulnerabilities and points of interest to the hackers with the goal of exposing the attacker prior to them finding an actual vulnerability on the site. Activity processors are the processors that monitor for and report any other malicious behavior. These operators watch for malicious activity based on non-injected points of interest. These typically involve monitoring headers, errors, input fields, URL sequences, etc, with the goal of identifying malicious behavior within the valid application stream.

Activity processors enable monitoring of session traffic. Things like authentication and cookies are among the types of traffic that become introspected by various activity processors.

Tracking processors, allow for more advanced tracking of the attackers. These processors attempt to collect additional data based on behavioral characteristics and unique attacker's environment information. These "fingerprints" become a basis for the "hacker database" used in detecting attackers from the first request they make.

Response processors are the processors that are used for generating response to the end user. If turned on, these can be used to either manually or automatically (depending on the configuration) respond to a hacker as soon as their activity is detected. In case of an automated response, these can be tuned to match more or less any condition including but not limited to frequency of occurrence, complexity, types of incidents triggered.

CHAPTER 29

Honeypot Processors

- [Honeypot Processors: Access Policy Processor on page 156](#)
- [Honeypot Processors: Access Policy Processor: Incidents - Malicious Service Call on page 157](#)
- [Honeypot Processors: Access Policy Processor: Incidents - Service Directory Indexing on page 157](#)
- [Honeypot Processors: Access Policy Processor: Incidents - Service Directory Spider on page 158](#)
- [Honeypot Processors: AJAX Processor on page 159](#)
- [Honeypot Processors: AJAX Processor: Incidents - Malicious Script Execution on page 160](#)
- [Honeypot Processors: AJAX Processor: Incidents - Malicious Script Introspection on page 161](#)
- [Honeypot Processors: Basic Authentication Processor on page 162](#)
- [Honeypot Processors: Basic Authentication Processor: Incidents - Apache Configuration Requested on page 163](#)
- [Honeypot Processors: Basic Authentication Processor: Incidents - Apache Password File Requested on page 164](#)
- [Honeypot Processors: Basic Authentication Processor: Incidents - Invalid Credentials on page 165](#)
- [Honeypot Processors: Basic Authentication Processor: Incidents - Protected Resource Requested on page 166](#)
- [Honeypot Processors: Basic Authentication Processor: Incidents - Password Cracked on page 166](#)
- [Honeypot Processors: Basic Authentication Processor: Incidents - Basic Authentication Brute Force on page 167](#)
- [Honeypot Processors: Cookie Processor on page 168](#)
- [Honeypot Processors: Cookie Processor: Incident - Cookie Parameter Manipulation on page 169](#)
- [Honeypot Processors: File Processor on page 170](#)
- [Honeypot Processors: File Processor: Incident - Suspicious Filename on page 170](#)
- [Honeypot Processors: File Processor: Incident - Suspicious File Exposed on page 171](#)

- [HoneyPot Processors: File Processor: Incident - Suspicious Resource Enumeration on page 172](#)
- [HoneyPot Processors: Hidden Input Form Processor on page 173](#)
- [HoneyPot Processors: Hidden Input Form Processor: Incident - Parameter Type Manipulation on page 173](#)
- [HoneyPot Processors: Hidden Input Form Processor: Incident - Hidden Parameter Manipulation on page 174](#)
- [HoneyPot Processors: Hidden Link Processor on page 175](#)
- [HoneyPot Processors: Hidden Link Processor: Incident - Link Directory Indexing on page 176](#)
- [HoneyPot Processors: Hidden Link Processor: Incident - Link Directory Spidering on page 176](#)
- [HoneyPot Processors: Hidden Link Processor: Incident - Malicious Resource Request on page 177](#)
- [HoneyPot Processors: Query String Processor on page 177](#)
- [HoneyPot Processors: Query String Processor: Incident - Query Parameter Manipulation on page 178](#)
- [HoneyPot Processors: Robots Processor on page 179](#)
- [HoneyPot Processors: Robot Processor: Incident - Malicious Spider Activity on page 179](#)

HoneyPot Processors: Access Policy Processor

This processor injects fake permission data into the clientaccesspolicy.xml file of the web application's domain. The fake access policy references a fake service and grants a random domain access to call it. If the service is ever called, or any files are ever requested in the directory the service is supposedly contained in, an incident can be created. Under normal conditions, no user will ever see the clientaccesspolicy.xml file, and therefore be unaware of the URL to the fake service or the directory it resides in. In the cases where a Silverlight object is legitimately requesting clientaccesspolicy.xml from the protected domain in order to access a known service, it will not create an incident, because the service being called is defined with real access directives.

Table 11: Access Policy Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether or not to enable this process for https traffic.
Advanced			
Fake Service	String	Random	The fake service the user requested.
Incident: Malicious Service Call	Boolean	True	The user manually entered the URL into the browser and accessed the service that way. They did not call the function.

Table 11: Access Policy Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Incident: Service Directory Indexing	Boolean	True	The user asked for a file index on the directory that contains the fake service.
Incident: Service Directory Spider	Boolean	True	The user is issuing requests for resources inside the directory that contains the fake service. Since the directory does not exist, all of these types of requests are unintended and malicious.

Honeypot Processors: Access Policy Processor: Incidents - Malicious Service Call

Complexity: Medium (3.0)

Default Response: 1x = 5 day Clear Inputs

Cause: WebApp Secure adds a fake cookie to the websites it protects. The cookie is intended to look as though it is part of the applications overall functionality, and is often selected to appear vulnerable (such as naming the cookie 'debug' or 'admin' and giving it a numerical or Boolean value). The "Cookie Parameter Manipulation" incident is triggered whenever the fake cookie value changes its value.

Behavior: Modifying the inputs of a page is the foundation of a large variety of attack vectors. Basically, if you want to get the backend server to do something different, you need to supply different input values (either by cookie, query string, url, or form parameters). Depending on what value the user chose for the input, the attack could fall under large number of vectors, including "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", and "SQL injection" among many others. A common practice is to first spider the web site, then test every single input on the site for a specific set of vulnerabilities. For example, the user might first index the site, then visit each page on the site, then test every exposed input (cookie, query string, and form inputs) with a list of SQL injection tests. These tests are designed to break the resulting page if the input is vulnerable. As such, the entire process (which can involve thousands of requests) can be automated and return a clean report on which inputs should be targeted. Because a WebApp Secure cookie looks just like a normal application cookie, a spider that tests all inputs will eventually test the fake cookie as well. This means that if there is a large volume of this incident, it is likely due to such an automated process. It should be assumed that the values tested against the fake cookie, have also been tested against the rest of the cookies on the site.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Honeypot Processors: Access Policy Processor: Incidents - Service Directory Indexing

Complexity: Medium (3.0)

Default Response: 1x = 5 day Block

Cause: Originally, embedded HTML technologies such as Flash and Java, were not able to communicate with 3rd party domains. This was a security constraint to prevent a malicious Java or Flash object from performing unwanted actions against a site other than the one hosting the object (for example, a Java applet that brute forces a Gmail login in the background). This limitation was eventually decreased in order to facilitate more complex mash-ups of information from a variety of sources. However to prevent any untrusted web sites from abusing this new capability, a resource called the "clientaccesspolicy.xml" was introduced. Now, when a plugin object wants to communicate with a different domain, it will first request "clientaccesspolicy.xml" from that domain. If the file specifies that the requesting domain is allowed to access the specified resource, then the plugin object will be given permission to communicate directly with the 3rd party. The clientaccesspolicy.xml therefore provides a convenient reference for hackers when trying to scope the attack surface of the web site. For example, there may be a vulnerable service listed in clientaccesspolicy.xml, but that service may not be referenced anywhere else on the site. So unless the hacker looks at clientaccesspolicy.xml, they would never even know the service existed. WebApp Secure will inject a fake service definition into the clientaccesspolicy.xml file in order to identify which users are manually probing the file for information. The "Service Directory Indexing" incident will be triggered if the user attempts to get a file listing from the directory the fake service is supposedly located in.

Behavior: Attempting to get a file listing from the directory where the potentially vulnerable service is located is likely in an effort to identify other unreferenced vulnerable services, or possibly even data or source files used by the service. Such a request represents a "Directory Indexing" attack, and is generally performed while attempting to establish a full understanding of a websites attack surface.

Honeypot Processors: Access Policy Processor: Incidents - Service Directory Spider

Complexity: Medium (3.0)

Default Response: 1x = 5 day Block

Cause: Originally, embedded HTML technologies such as Flash and Java, were not able to communicate with 3rd party domains. This was a security constraint to prevent a malicious Java or Flash object from performing unwanted actions against a site other than the one hosting the object (for example, a Java applet that brute forces a Gmail login in the background). This limitation was eventually decreased in order to facilitate more complex mash-ups of information from a variety of sources. However to prevent any untrusted websites from abusing this new capability, a resource called the "clientaccesspolicy.xml" was introduced. Now, when a plugin object wants to communicate with a different domain, it will first request "clientaccesspolicy.xml" from that domain. If the file specifies that the requesting domain is allowed to access the specified resource, then the plugin object will be given permission to communicate directly with the 3rd party. The clientaccesspolicy.xml therefore provides a convenient reference for hackers when trying to scope the attack surface of the web site. For example, there may be a vulnerable service listed in clientaccesspolicy.xml, but that service may not be referenced anywhere else on the site. So unless the hacker looks at clientaccesspolicy.xml,

they would never even know the service existed. WebApp Secure will inject a fake service definition into the `clientaccesspolicy.xml` file in order to identify which users are manually probing the file for information. The "Service Directory Spidering" incident will be triggered if the user attempts to request a random file inside the directory the fake service is supposedly located in.

Behavior: Requesting a random file from the directory where the potentially vulnerable service is supposedly located is likely in an effort to identify other unreferenced resources. This could include configuration files, other services, data files, etc. Usually an attacker will first attempt to get a full directory index (which only takes one request), but if that fails, the only other technique is to guess the filenames (which could take thousands of requests). Because guessing the file names can take so many requests, there are several publicly available tools that can enumerate over a large list of common file and directory names in a matter of minutes. This type of behavior is an attempt to exploit a server for "Predictable Resource Location" vulnerabilities, and is generally done while the attack is trying to scope the web applications attack surface.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium Web Site](#) and search for the attack name to learn more about it.

Honeypot Processors: AJAX Processor

A mistake commonly made by web developers is to consolidate every JavaScript file used by their web site into a single file. They then reference that one file from every page on the site, regardless of whether it needs all of the code defined in the file. This is an optimization trick that works, but exposes potential vulnerabilities. The goal is to get the browser to cache all of the external JavaScript, so that you don't need to keep downloading additional code as you navigate the site. Consider the case where one of the pages on the site contains an administrative console written with AJAX technology. In the administrative page, there is a JavaScript file that contains code for managing users of the site (creating user, deleting users, getting user details, etc). Normally only administrators would visit this page, and they would be the only ones who can see this code. Once all JavaScript on the site is consolidated however, these types of sensitive functions tend to get mixed into the rest of the safer functions. Hackers look for these types of functions in order to find both the administrative page that uses them, as well as exploit the function itself. The goal of this trap is to emulate this common mistake and entice hackers into attempting to exploit the "sensitive looking" function.

Table 12: AJAX Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.

Table 12: AJAX Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Advanced			
Inject Script Enabled	Boolean	True	Whether to inject the fake Javascript code into HTML responses.
Service	Configurable	AJAX Service	The fake service to expose.
Incident: Malicious Script Execution	Boolean	True	The user executed the fake JavaScript function.
Incident: Malicious Script Introspection	Boolean	True	The user manually entered the URL into the browser and accessed the service that way. They did not call the function.

Honeypot Processors: AJAX Processor: Incidents - Malicious Script Execution

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection 2-6 seconds and permanent Clear Inputs in 10 minutes.

Cause: WebApp Secure injects a fake JavaScript file into the websites it protects. This fake JavaScript file is designed to look as though it is intended for administrative use only, but has been mistakenly linked in with non administrative pages. The JavaScript file exposes an AJAX function that communicates with a potentially vulnerable fake service. If the user attempts to invoke this function using a tool like Firebug, this incident will be triggered.

Behavior: It is common practice to create a few single JavaScript files that contain the majority of the code your site needs, and then importing that code into all of the pages. This increases the performance of the site, because the user can download and cache all the JavaScript at once, rather than having to re-download all or some of it again on every page change. However in some cases, developers mistakenly include sensitive administrative functions in with common functions needed by unauthenticated users. For example, a developer might include an "addUser" function into a file that also contains a "changeImageOnHover" function. The "addUser" function may only be called from an administrative UI (behind a login), while the hover image effect would be called on a lot of different pages. Hackers often look through all of the various Javascript files being included on the pages of a website in order to find references to other services that might be vulnerable. Once a function has been identified, the hacker will attempt to find a way to exploit the service the function uses. Because the attacker is actually executing the function instead of attempting to directly communicate with the potentially vulnerable service, this is likely a less sophisticated attack. They are more than likely just trying to determine if the service actually exists, and if they can call it without being authenticated, however depending on the values they supplied as arguments to the function, this could be a number of different attack types, including "Abuse of Functionality", "Buffer

Overflow", "Denial of Service", "Format String", "Integer Overflows", "OS Commanding", and "SQL Injection."



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Honeypot Processors: AJAX Processor: Incidents - Malicious Script Introspection

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection 2-6 seconds and Captcha. 2x = Slow Connection 4-14 seconds and permanent Block in 10 minutes.

Cause: WebApp Secure injects a fake JavaScript file into the websites it protects. This fake JavaScript file is designed to look as though it is intended for administrative use only, but has been mistakenly linked in with non administrative pages. The JavaScript file exposes an AJAX function that communicates with a potentially vulnerable fake service. If the user manually inspects the code of the function and attempts to exploit the service it uses directly (without calling the function itself), this incident will be triggered.

Behavior: To improve performance of a web site, by minimizing the number of HTTP requests (and taking advantage of browser-side caching), web developers commonly combine most of their JavaScript code into just a few files, which are then included in the HTML of the entire site. However, in some cases, developers mistakenly include sensitive administrative functions in with common functions needed by unauthenticated users. For example, a developer might include an "addUser" function into a file that also contains a "changeImageOnHover" function. The "addUser" function may only be called from an administrative UI (behind a login), while the hover image effect would be called on a lot of different pages. Hackers often look through all of the various Javascript files being included on the pages of a website in order to find references to other services that might be vulnerable. Once a function has been identified, the hacker will attempt to find a way to exploit the service the function uses. Unlike the malicious script execution incident, here the attacker has actually dissected the fake AJAX function and attempted to directly exploit the service it uses. This is a more sophisticated attack than actually calling the Javascript function, because it requires that the user understand Javascript logic. Depending on what values they are sending to the service, this could be in an effort to perform any number of exploits, including Abuse of Functionality, "Buffer Overflow", "Denial of Service", "Format String", "Integer Overflows", "OS Commanding", and "SQL Injection."



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

HoneyPot Processors: Basic Authentication Processor

The basic authentication processor is responsible for emulating a vulnerable authentication mechanism in the web application. This is done by publicly exposing fake server configuration files (.htaccess and .htpasswd) that appear to be protecting a resource with basic authentication (a part of the HTTP protocol). To the attacker, the site will appear to be exposing a sensitive administrative script on the site, with weak password protection. As the malicious user identifies the availability of such publicly exposed files, they are walked through a series of steps that emulate exposing an additional piece of information. As the final step, if they end up breaking the weakly authenticated password, they will be considered a high threat.



NOTE: This processor should only be used when the site is using Apache as front end web servers due to particular files involved (.htaccess and .htpasswd) being specific to Apache web server.)



NOTE: Browsers often ignore the body content of HTTP responses if the status code is anything other than 200. For best compatibility with different browser versions, you may wish to use a 200 status code when uploading responses such as images or executable code.

Table 13: Basic Authentication Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			Whether traffic should be passed through this processor.
Processor Enabled	Boolean	True	
Advanced			
Authorized Users	Collection	Collection	A list of authorized user accounts.
Protected Resource URL	String	[random resource]	The fake protected resource.
Protected Resource Response Status	String	[random status]	The HTTP status to return when accessing the resource.
Randomization Salt	String	Random	A random set of characters used to salt the generation of code. Any value is fine here.
Incident: Password Cracked	Boolean	True	The user has successfully accessed a fake protected resource using a cracked username and password.

Table 13: Basic Authentication Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Incident: Apache Configuration Requested	Boolean	True	The user has requested the apache directory configuration file .htaccess.
Incident: Apache Password File Requested	Boolean	True	The user has requested the apache password file .htpasswd
Incident: Invalid Credentials	Boolean	True	The user has attempted to login to access the fake file protected by basic authentication, but failed.
Incident: Protected Resource Requested	Boolean	True	The user has requested a fake file which is protected by basic authentication.

Honeypot Processors: Basic Authentication Processor: Incidents - Apache Configuration Requested

Complexity: Low (2.0)

Default Response: none.

Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given website is probably running apache. One such vulnerability involves the use of an .htaccess22 file to provide directory level configuration (such as default 404 messages, password protected resources, directory indexing options, etc...), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources. In such a scenario, a hacker could gain valuable intelligence on the way the server is configured.

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration" weakness that might expose a "Credential/Session Prediction", "OS Commanding", "Path Traversal", or "URL Redirector Abuse" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is related to .htaccess. They are requesting a user database file for a password protected resource defined in .htaccess. This file is generally named ".htpasswd". The user either opened the .htaccess file and found the reference to .htpasswd, or they simply tried .htpasswd to see if anything came back (with or without asking for .htaccess). Either way, this behavior is involved in the establishment of a "Credential/Session Prediction" vulnerability. The request for .htpasswd is usually performed while attempting to establish the scope of the websites attack surface, although sometimes is not performed until trying to identify a valid attack vector.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Honeypot Processors: Basic Authentication Processor: Incidents - Apache Password File Requested

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds.

Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given website is probably running apache. One such vulnerability involves the use of an .htaccess28 file to provide directory level configuration (such as default 404 messages, password protected resources, directory indexing options, etc...), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources. In such a scenario, a hacker could gain valuable intelligence on the way the server is configured. WebApp Secure will automatically block any requests for the .htaccess resource, and return a fake version of the file. The fake version of the file will contain the directives necessary to password protect a fake resource. These directives allude to the existence of a user database file that contains usernames and encrypted passwords. The "Apache Password File Requested" incident will trigger in the event that the user requests the fake user database file (generally named .htpasswd).

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration" weakness that might expose a "Credential/Session Prediction", "OS Commanding", "Path Traversal", or "URL Redirector Abuse" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is related to .htaccess. They are requesting a user database file for a password protected resource defined in .htaccess. This file is generally named ".htpasswd". The user either opened the .htaccess file and found the reference to .htpasswd, or they simply tried .htpasswd to see if anything came back (with or without asking for .htaccess). Either way, this behavior is involved in the establishment of a "Credential/Session Prediction" vulnerability. The request for .htpasswd is usually performed while attempting to establish the scope of the websites attack surface, although sometimes is not performed until trying to identify a valid attack vector.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Honeypot Processors: Basic Authentication Processor: Incidents - Invalid Credentials

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection 2-6 seconds. 15x = Basic Authentication Bruteforce Incident.

Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given web site is probably running apache. One such vulnerability involves the use of an .htaccess file to provide directory level configuration (such as default 404 messages, password protected resources, directory indexing options, etc), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources. In such a scenario, a hacker could gain valuable intelligence on the way the server is configured. WebApp Secure will automatically block any requests for the .htaccess resource, and return a fake version of the file. The fake version of the file will contain the directives necessary to password protect a fake resource. Should the user request the password protected resource, WebApp Secure will simulate the correct authentication method defined in .htaccess, and simulate the existence of the fake resource. The "Invalid Credentials" incident will trigger in the event that the user requests the fake password protected file and supplies an invalid username and password (as would be the case if they requested the file in a browser and guessed a username and password at the login prompt).

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration" weakness that might expose a "Credential/Session Prediction", "OS Commanding", "Path Traversal", or "URL Redirector Abuse" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is referenced only from .htaccess. The fake resource is password protected, and the user has attempted to authenticate with bad credentials. This is most likely in an effort to guess a valid username and password combination, such as "admin:admin", or "guest:guest". It may also be part of a larger brute force attempt, where the attacker tries a long list of possible combinations. This is a poor method for locating valid usernames and passwords, especially since the user database file .htpasswd is actually exposed (albeit fake). So a brute force attack (represented by a large quantity of this incident type) generally means the attacker is less sophisticated.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

HoneyPot Processors: Basic Authentication Processor: Incidents - Protected Resource Requested

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds.

Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given website is probably running apache. One such vulnerability involves the use of an .htaccess file to provide directory level configuration (such as default 404 messages, password protected resources, directory indexing options, etc), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources. In such a scenario, a hacker could gain valuable intelligence on the way the server is configured. WebApp Secure will automatically block any requests for the .htaccess resource, and return a fake version of the file. The fake version of the file will contain the directives necessary to password protect a fake resource. Should the user request the password protected resource, WebApp Secure will simulate the correct authentication method defined in .htaccess, and simulate the existence of the fake resource. The "Protected Resource Requested" incident will trigger in the event that the user requests the fake password protected file and does not supply a username and password (as would be the case if they requested the file in a browser and canceled the login prompt).

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration" weakness that might expose a "Credential/Session Prediction", "OS Commanding", "Path Traversal", or "URL Redirector Abuse" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is referenced only from .htaccess. The resource is password protected, but the user has not yet tried to supply credentials. This is most likely in an attempt to see if the password protected file actually exists.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

HoneyPot Processors: Basic Authentication Processor: Incidents - Password Cracked

Complexity: High (4.0)

Default Response: 1x = Permanent Block.

Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given website is probably running apache. One such vulnerability involves the use of an .htaccess file to provide directory level configuration (such as default 404 messages, password protected resources, directory indexing options, etc.), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources. In such a scenario, a hacker could gain valuable intelligence on the way the server is configured. WebApp Secure will automatically block any requests for the .htaccess resource, and return a fake version of the file. The fake version of the file will contain the directives necessary to password protect a fake resource. The directives will also allude to the existence of a password database file. If the attacker requests the password database file, and then uses a tool such John The Ripper to crack one of the encrypted passwords, they will be able to authenticate against the fake protected resource successfully. Should the user request the password protected resource, and supply a valid username and password combination (as defined in the password database), the "Password Cracked" incident will be triggered.

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration" weakness that might expose a "Credential/Session Prediction", "OS Commanding", "Path Traversal", or "URL Redirector Abuse" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is referenced only from .htaccess. The fake resource is password protected, and the user has supplied valid authentication credentials. The only way to obtain valid credentials is to either brute force the login (which would be the case if there were excessive numbers of "Invalid Credential" incidents), or to access the fake password database file (usually .htpasswd) and crack one of the encrypted passwords using an encryption cracking tool. This represents the final and most complicated step in a successful "Credential/Session Prediction" exploit, and is usually performed long after the attack surface of the site has been fully scoped. Unless there are excessive numbers of "Invalid Credential" incidents, which would be the case in a brute force attack, the user must have also requested ".htpasswd", and therefore should also have an "Apache Password File Requested" incident. If this incident is missing, then the hacker has likely established two independent profiles in WebApp Secure.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium Web Site](#) and search for the attack name to learn more about it.

Honeypot Processors: Basic Authentication Processor: Incidents - Basic Authentication Brute Force

Complexity: Medium (3.0)

Default Response: 1X - CAPTCHA; 2x = Permanent Block.

Cause: Apache is a very common web server. As a result, hackers will often look for vulnerabilities specific to Apache, since there is a good chance that any given website is running Apache. One such vulnerability involves the use of an .htaccess file to provide directory-level configuration (password-protected resources, directory indexing options, etc), while not sufficiently protecting the .htaccess file itself. By convention, configuration files should not be exposed to the public — so if a user requests .htaccess or a related resource, they should get either a "404 Not Found" or "403 Forbidden" error. Unfortunately, an improperly-configured installation of Apache may not block requests for these resources. In such a scenario, a hacker could gain valuable knowledge of the way the server is configured. WebApp Secure will automatically block any requests for the .htaccess resource, and instead return a fake version of the file, which contains the directives necessary to password-protect a fake resource. Should the user request the password-protected resource, WebApp Secure will simulate the correct authentication method defined in .htaccess, and simulate the existence of the fake resource. The "Basic Authentication Brute Force" incident will trigger in the event that the user requests the fake passwordprotected file and repeatedly supplies an invalid username and password (as would be the case if the user were guessing various username and password combinations).

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration" weakness that might expose a "Credential/Session Prediction", "OS Commanding", "Path Traversal", or "URL Redirector Abuse" vulnerability among others.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

In this specific case, the attacker is requesting a different resource that is referenced only from .htaccess. The fake resource is password-protected, and the user has attempted to authenticate with a large number of bad credentials. This is most likely in an effort to guess a valid username and password combination, such as "admin:admin", or "guest:guest". This is a poor method for locating valid usernames and passwords, especially since the user database file .htpasswd is actually exposed (albeit fake). So a brute force attack generally means the attacker is less sophisticated. Because the password-protected file is not referenced from anywhere outside of .htaccess, this incident should not happen unless an "Apache Configuration Requested" incident has occurred first. If that is not the case, then the hacker has likely established two independent profiles in WebApp Secure. This type of behavior is generally performed when attempting to establish a successful attack vector.

Honeypot Processors: Cookie Processor

Cookies are used by web applications to maintain state for a given user. They consist of key/value pairs that are passed around in headers and also stored client side. Each

key/value pair has various attributes including which domains it is valid for, what paths within those domains, as well as security restrictions and expiration information. Because this is the primary way for a web application to maintain a session, hackers will often try to manipulate cookie values manually in an effort to escalate access or hijack someone else's session. All of the attacks applicable to modifying form parameters are also applicable to modifying cookie parameters. It may even be possible, although unlikely, to find an SQL injection flaw in a cookie parameter.

Table 14: Cookie Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether or not to enable this process for http traffic.
Advanced			
Cookie	String	Cookie	The fake cookie to use.
Incident: Cookie Parameter Manipulation	Boolean	True	The user modified the value of a cookie which should never be modified.

Honeypot Processors: Cookie Processor: Incident - Cookie Parameter Manipulation

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection 2-6 seconds and permanent Clear Inputs in 10 minutes.

Cause: WebApp Secure adds a fake cookie to the web sites it protects. The cookie is intended to look as though it is part of the applications overall functionality, and is often selected to appear vulnerable (such as naming the cookie 'debug' or 'admin' and giving it a numerical or Boolean value). The "Cookie Parameter Manipulation" incident is triggered whenever the fake cookie value changes its value.

Behavior: Modifying the inputs of a page is the foundation of a large variety of attack vectors. Basically, if you want to get the backend server to do something different, you need to supply different input values (either by cookie, query string, url, or form parameters). Depending on what value the user chose for the input, the attack could fall under large number of vectors, including "Buffer Overflow", "XSS5", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", and "SQL injection" among many others. A common practice is to first spider the website, then test every single input on the site for a specific set of vulnerabilities. For example, the user might first index the site, then visit each page on the site, then test every exposed input (cookie, query string, and form inputs) with a list of SQL injection tests. These tests are designed to break the resulting page if the input is vulnerable. As such, the entire process (which can involve thousands of requests) can be automated and return a clean report on which inputs should be targeted. Because WebApp Secure cookie looks just

like a normal application cookie, a spider that tests all inputs will eventually test the fake cookie as well. This means that if there is a large volume of this incident, it is likely due to such an automated process. It should be assumed that the values tested against the fake cookie, have also been tested against the rest of the cookies on the site.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Honeypot Processors: File Processor

When developing websites, administrators will often rename files in order to make room for a newer version of the file. They may also archive older files. A common vulnerability is the case where these older files are left in the web accessible directories, and they contain non static resources. For example, consider the case where a developer renames shopping_cart.php to shopping_cart.php.bak. If an attacker looks for php files and tries to access all of them with a .bak extension, they may stumble across the backup file. Because the server is not configured to parse .bak files as php files, it will serve the unexecuted script source code to the client. This technique can yield database credentials, system credentials, as well as expose more serious vulnerabilities in the code itself. The goal of this processor is to detect when a user is attempting to find unreferenced files.

Table 15: File Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Block Response	Configurable	HTTP Response	The response to return when a request is blocked due to a matching suspicious token rule with blocking enabled.
Suspicious Tokens	Collection	Collection	The configured suspicious extensions.
Incident: Suspicious File Exposed	Boolean	True	A file which has a suspicious filename is publicly available.
Incident: Suspicious Filename	Boolean	True	A file with a filename that contains a suspicious token was requested.

Honeypot Processors: File Processor: Incident - Suspicious Filename

Complexity: Suspicious (1.0)

Default Response: 10x = Suspicious Resource Enumeration Incident.

Cause: WebApp Secure has a list of file tokens which represent potentially sensitive files. For example, developers will often rename source files with a ".bck" extension during debugging, and sometimes they forget to delete the backup after they are done. Hackers often look for these left over source files. WebApp Secure is configured to look for any request to a file with a ".bck" extension (as well as any other configured extensions), and trigger this incident if the file does not exist. An incident will not be triggered if the file does in fact exist, and the extension is not configured to block the response. This is to avoid legitimate files being flagged as suspicious filenames.

Behavior: There are specific files that many websites host, that contain valuable information for a hacker. These files generally include data such as passwords, SQL schema's, source code, etc. When hackers try to breach a site, they will often check to see if they can locate some of these special files in order to make their jobs easier. For example, if a hacker sees that the home page is called "index.php", they may try and request "index.php.bak", because if it exists, it will be returned as raw source code. This is usually an effort to exploit a "Predictable Resource Location" vulnerability. Automated scanners will generally test all of these types of extensions (.bck, .bak, .zip, .tar, .gz, etc...) against every legitimate file that is located through simple spidering. Because this incident is only created if the file being requested does not actually exist, it does not represent a successful exploit.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Honeypot Processors: File Processor: Incident - Suspicious File Exposed

Complexity: Suspicious (1.0)

Default Response: 10x = Suspicious Resource Enumeration Incident.

Cause: WebApp Secure has a list of file tokens which represent potentially sensitive files. For example, developers will often rename source files with a ".bck" extension during debugging, and sometimes they forget to delete the backup after they are done. Hackers often look for these left over source files. WebApp Secure is configured to look for any request to a file with a ".bck" extension (as well as any other configured extensions), and trigger this incident if the extension is configured as illegal. This incident will only be triggered if the file actually exists, and the request reaches the backend server. For example, the user might request "database.sql". If the .sql extension is configured to block, and the file actually exists on the server, this incident will be generated. If "database.sql" does not exist, then only a "Suspicious Filename" incident will be created.

Behavior: There are specific files that many web sites host, that contain valuable information for a hacker. These files generally include data such as passwords, SQL schema's, source code, etc. When hackers try to breach a site, they will often check to see if they can locate some of these special files in order to make their jobs easier. For

example, if a hacker sees that the home page is called "index.php", they may try and request "index.php.bak", because if it exists, it will be returned as raw source code. This is usually an effort to exploit a "Predictable Resource Location" vulnerability. Automated scanners will generally test all of these types of extensions (.bck, .bak, .zip, .tar, .gz, etc...) against every legitimate file that is located through simple spidering. This incident is only triggered when the user requested a file that would otherwise have been successfully returned, if it were not blocked by WebApp Secure. For example, the user might request "database.sql" and actually get a 200 response from the server indicating that the file exists and is accessible to everyone. However if the system is configured to mark the ".sql" extension as illegal, then WebApp Secure will block the request. This prevents the sensitive file from potentially being exposed to an actual malicious user. If this incident occurs, the server administrator should immediately remove the sensitive file or change its permissions so it is no longer publicly accessible.

Honeypot Processors: File Processor: Incident - Suspicious Resource Enumeration

Complexity: Low (2.0)

Default Response: 1x = 5 day Block.

Cause: WebApp Secure has a list of file tokens which represent potentially sensitive files. For example, developers will often rename source files with a ".bck" extension during debugging, and sometimes they forget to delete the backup after they are done. Hackers often look for these left over source files. WebApp Secure is configured to look for any request to a file with a ".bck" extension (as well as any other configured extensions), and trigger a Suspicious Filename incident if the file does not exist. Should the suspicious filename incident be triggered several times, this incident will then be triggered.

Behavior: There are specific files that many web sites host, that contain valuable information for a hacker. These files generally include data such as passwords, SQL schema's, source code, etc... When hackers try to breach a site, they will often check to see if they can locate some of these special files in order to make their jobs easier. For example, if a hacker sees that the home page is called "index.php", they may try and request "index.php.bak", because if it exists, it will be returned as raw source code. This is usually an effort to exploit a "Predictable Resource Location" vulnerability. Automated scanners will generally test all of these types of extensions (.bck, .bak, .zip, .tar, .gz, etc...) against every legitimate file that is located through simple spidering. The first few times a user requests a filename containing a suspicious token, they will only get "Suspicious Filename" incidents. However if they request a large volume of filenames with suspicious tokens, then the "Suspicious Resource Enumeration" incident is generated. This incident represents a user who is actively scanning the site with very aggressive tactics to find unlinked and sensitive data.



NOTE: For information on the attack types mentioned here, go to [The Web Application Security Consortium Web Site](#) and search for the attack name to learn more about it.

Honeypot Processors: Hidden Input Form Processor

Many webmasters create forms which post to a common form handling service; using hidden fields to indicate how the service should handle the data. A common hacking technique is to look for these hidden parameters and see if there is any way to change the behavior of the service by manipulating its input parameters. This processor is responsible for injecting a fake hidden input into forms in HTML responses and ensuring that when those values are posted back to the server, they have not been modified.

Table 16: Hidden Input Form Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Hidden Input Parameter	Collection	Collection	The possible hidden inputs on a page.
Inject Input Enabled	Boolean	True	Whether to inject hidden inputs into HTML forms.
Maximum Injections	Integer	3	The maximum number of fake hidden parameters that will be added to any given URL.
Strip Fake Input	Boolean	True	Whether to remove the fake input value from the posted form results before proxying the request to the backend servers. This should only be turned off if there is some additional security implemented on the form, where its contents are signed on the client and validated on the server.
Incident: Hidden Parameter Manipulation	Boolean	True	The user submitted the form and the value of the injected parameter is not what was expected.
Incident: Hidden Input Type Manipulation	Boolean	True	The user submitted the form and the value of the injected parameter is not what was expected. It was also modified to post a file.

Honeypot Processors: Hidden Input Form Processor: Incident - Parameter Type Manipulation

Complexity: High (4.0)

Default Response: 1x = Permanent Clear Inputs.

Cause: WebApp Secure inspects outgoing traffic for HTML forms with a "POST" method type. Forms that post to a local URL (within the same domain), will be modified to include a fake hidden input with a defined value. The input is intended to look as though it was always part of the form, and is often selected to appear vulnerable (such as naming the

input 'debug' or 'loglevel' and giving it a numerical or Boolean value). The input will however, always be assigned a value that can be represented as a string of characters (in other words, not binary data). The "Parameter Type Manipulation" incident is triggered whenever the fake hidden input is modified from its originally assigned value in order to submit a multipart file.

Behavior: Modifying the inputs of a page is the foundation of a large variety of attack vectors. Basically, if you want to get the backend server to do something different, you need to supply different input values (either by cookie, query string, url, or form parameters). Depending on what value the user chose for the input, the attack could fall under large number of vectors, including "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", and "SQL injection" among many others. Unlike a normal "Hidden Parameter Manipulation" incident, this version is triggered when the user changes the encoding of the form and submits the hidden input as a file post. This is likely in an attempt to either achieve a "Buffer Overflow", or to exploit a filter evasion weakness, that might have otherwise blocked the value being submitted. A common practice is to first spider the website, then test every single input on the site for a specific set of vulnerabilities. For example, the user might first index the site, then visit each page on the site, then test every exposed input (cookie, query string, and form inputs) with a list of SQL injection tests. These tests are designed to break the resulting page if the input is vulnerable. As such, the entire process (which can involve thousands of requests) can be automated and return a clean report on which inputs should be targeted. Because WebApp Secure injects several fake inputs, a spider that tests all inputs will eventually test the fake input as well. This means that if there is a large volume of this incident, it is likely due to such an automated process. It should be assumed that the values tested against the fake input, have also been tested against the rest of the inputs on the site.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Honeypot Processors: Hidden Input Form Processor: Incident - Hidden Parameter Manipulation

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection 2-6 seconds. 2x = Logout User. 3x = Clear Inputs.

Cause: WebApp Secure inspects outgoing traffic for HTML forms with a "POST" method type. Forms that post to a local URL (within the same domain), will be modified to include a fake hidden input with a defined value. The input is intended to look as though it was always part of the form, and is often selected to appear vulnerable (such as naming the input 'debug' or 'loglevel' and giving it a numerical or Boolean value). The "Hidden Parameter Manipulation" incident is triggered whenever the fake hidden input is modified from its originally assigned value.

Behavior: Modifying the inputs of a page is the foundation of a large variety of attack vectors. Basically, if you want to get the backend server to do something different, you need to supply different input values (either by cookie, query string, url, or form parameters). Depending on what value the user chose for the input, the attack could fall under large number of vectors, including "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", and "SQL injection" among many others. A common practice is to first spider the website, then test every single input on the site for a specific set of vulnerabilities. For example, the user might first index the site, then visit each page on the site, then test every exposed input (cookie, query string, and form inputs) with a list of SQL injection tests. These tests are designed to break the resulting page if the input is vulnerable. As such, the entire process (which can involve thousands of requests) can be automated and return a clean report on which inputs should be targeted. Because WebApp Secure injects several fake inputs, a spider that tests all inputs will eventually test the fake input as well. This means that if there is a large volume of this incident, it is likely due to such an automated process. It should be assumed that the values tested against the fake input, have also been tested against the rest of the inputs on the site.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Honeypot Processors: Hidden Link Processor

When trying to exploit a site, hackers will often scan the contents of the site in search of directories and files that are of interest. Because this activity is done at the source level, the hacker finds every file referenced, whereas when a user views a web site, they can only see the links that are visible according to the HTML. This processor injects a fake link into documents that references a file that looks interesting. The link is injected in such a way that prevents it from being rendered when the browser loads the page. This means that no normal user would ever find/click on the link, but that a scanner or hacker who is looking at the source code likely will.

Table 17: Hidden Link Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Hidden Links	Configurable	Hidden Links	The set of hidden links that can be injected into the site.
Inject Link Enabled	Boolean	True	Whether to inject the link into HTTP responses.

Table 17: Hidden Link Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Incident: Link Directory Indexing	Boolean	True	The user requested a directory index on one of the fake parent directories of the linked file.
Incident: Link Directory Spidering	Boolean	True	The user requested a resource inside the fake directory of the linked file.
Incident: Malicious Resource Request	Boolean	True	The user requested the fake linked resource.

Honeypot Processors: Hidden Link Processor: Incident - Link Directory Indexing

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds and 1 day Block.

Cause: WebApp Secure injects a hidden link into pages on the protected web application. This link is not exposed visually to users of the website. In order to find the link, a user would need to manually inspect the source code of the page. If a user finds the hidden link code in the HTML, and attempts to get a directory file listing from the directory the link points to, this incident will be triggered.

Behavior: A common technique for hackers when scoping the attack surface of a website is to spider the site and collect the locations of all of its pages. This is generally done using a simple script that looks for URL's in the returned HTML of the home page, then requests those pages and checks for URL's in their source, and so forth. Legitimate search engine spiders will do this as well. But the difference between a legitimate spider and a malicious user, is how aggressively they will use the newly discovered URL to derive other URLs. This incident triggers when the user goes beyond just checking the linked URL, but instead also attempts to get a file listing from the directory the URL points to. A legitimate spider would not do this, because it is considered fairly invasive. This activity is generally looking for a "Directory Indexing" weakness on the server, in an effort to locate unlinked and possibly sensitive resources.

Honeypot Processors: Hidden Link Processor: Incident - Link Directory Spidering

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds and 5 day Block in 6 minutes.

Cause: WebApp Secure injects a hidden link into pages on the protected web application. This link is not exposed visually to users of the website. In order to find the link, a user would need to manually inspect the source code of the page. If a user finds the hidden link code in the HTML, and attempts to request some other arbitrary file in the same fake directory as the link, this incident will be triggered.

Behavior: A common technique for hackers when scoping the attack surface of a website is to spider the site and collect the locations of all of its pages. This is generally done

using a simple script that looks for URL's in the returned HTML of the home page, then requests those pages and checks for URL's in their source, and so forth. Legitimate search engine spiders will do this as well. But the difference between a legitimate spider and a malicious user, is how aggressively they will use the newly discovered URL to derive other URLs. This incident triggers when the user goes beyond just checking the linked URL, but instead also attempts to request one or more arbitrary files inside the same directory as the file referenced by the hidden link. A legitimate spider would not do this, because it is considered fairly invasive. This activity is generally looking for a "Directory Indexing" weakness on the server, or a "Predictable Resource Location" vulnerability, in an effort to locate unlinked and possibly sensitive resources.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Honeypot Processors: Hidden Link Processor: Incident - Malicious Resource Request

Complexity: Suspicious (1.0)

Default Response: 1x = Slow Connection 2-6 seconds and 5 day Block in 6 minutes.

Cause: WebApp Secure injects a hidden link into pages on the protected web application, which is only discoverable through manual source code inspection. If a user discovers the hidden link, and attempts to request the file it references, this incident will be triggered.

Behavior: When scoping the attack surface of a web site, hackers commonly spider the site and collect the locations of all pages. Spidering can be performed with the assistance of simple scripts that look for URLs in the returned HTML of the home page, then request those pages and check for URLs in their source, and so forth. Legitimate search engine spiders will do this as well — but the difference between legitimate spiders and malicious users lies in how aggressively they will use the newly discovered URL to derive other URLs. This incident triggers when the user simply requests the hidden link URL. Because this can also be triggered by a legitimate search engine spider, this type of incident is not considered malicious on its own.

Honeypot Processors: Query String Processor

Hackers tend to manipulate the values of query string parameters in order to get the application to behave differently. The goal of this processor is to add fake query string parameters to some of the links and forms in the page, and verify that they do not get modified when accessed by the user.

Table 18: Query String Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			

Parameter	Type	Default Value	Description
-----------	------	---------------	-------------

Basic

Table 18: Query String Processor Configuration Parameters Parameter Type Default Value Description (*continued*)

Parameter	Type	Default Value	Description
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Fake Parameters	Collection	Collection	The collection of fake parameters to add to the links which already have parameters.
Inject Parameter Enabled	Boolean	True	Whether to inject query string parameters on urls in HTTP responses.
Maximum Injections	Integer	3	Whether to inject query string parameters on urls in HTTP responses.
Randomization Token	String	[Not Set]	Some web sites use complex redirection rules or modify query string parameters of static links using javascript on the client. In these situations, the randomization of fake query parameter values may be problematic. To resolve the issue, you can either update the list of fake parameters so that it does not include randomized tokens, or you can define a randomization token name here. If you define a randomization token, then the data used to randomize which value is selected will be transferred as an additional query string parameter by this name. It is recommended that you leave this field empty unless you experience a lot of fake positives on query parameter manipulation incidents shortly after setting up Webapp Secure to protect a website.
Strip Fake Input	Boolean	True	Whether to remove the fake input value from the query string before proxying the request to the backend servers. This should only be turned off if there is some additional security implemented on the site, where links are signed on the client and validated on the server.
Incident: Query Parameter Manipulation	Boolean	True	The user manually modified the value of a query string parameter.

HoneyPot Processors: Query String Processor: Incident - Query Parameter Manipulation

Complexity: Low (2.0)

Default Response: 3x = Slow Connection 2-6 seconds. 5x = 1 day Clear Inputs.

Cause: WebApp Secure injects a fake query parameter into some of the links of the protected web site. This query parameter has a known value, and should never change, because it is not part of the actual web application. If a user modifies the query parameter value, this incident will be triggered.

Behavior: Query parameters represent the most visible form of user input a web application exposes. They are clearly visible in the address bar, and can be easily changed by even an inexperienced user. However most users do not attempt to change values directly in the query string, unless they are trying to perform some action the website does not

normally expose through its interface, or does not make sufficiently easy. Because it is so easy for a normal user to accidentally change a query parameter, this incident alone is not considered strictly malicious. However depending on the value that is submitted, this could be part of a number of different exploit attempts, including "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", and "SQL injection".



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Honeypot Processors: Robots Processor

The Robots.txt proxy processor is responsible for catching malicious spiders that do not behave in accordance with established standards for spidering. Hackers often utilize the extra information sites expose to spiders, and then use that information to access resources normally not linked from the public site. Because this activity is effectively breaking established standards for spidering, this processor will also identify hackers who are using the information maliciously.

Table 19: Robots Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled Boolean	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Fake Disallowed Directories	String	Random	The path to a fake directory to add to the disallow rules in the robots.txt file. This path should be completely fake and not overlap with actual directories.
Incident: Malicious Spider Activity	Boolean	True	The user requested a resource which was restricted in the spider rules file, indicating this user is not a good spider, but is spidering the site anyway.

Honeypot Processors: Robot Processor: Incident - Malicious Spider Activity

Complexity: Low (2.0)

Default Response: 1x = Captcha and Slow Connection 2-6 seconds. 6x = 1 day Block.

Cause: One of the standard resources that just about every website should expose is called robots.txt. This resource is used by search engines to instruct them on how to spider the website. Two of the more important directives are "allow" and "disallow". These directives are used to identify which directories a spider should index, and which directories it should stay away from. Good practice for any website is to lock down any

resource that should not be exposed. However some web masters simply add a "disallow" statement so that those resources do not get indexed and therefore are never found by users. This technique does not work, because attackers will often access robots.txt and intentionally traverse the "disallow" directories in search of vulnerabilities. So in effect, the listing of such directories is basically pointing hackers in the direction of the most sensitive resources on the site. WebApp Secure will intercept requests for robots.txt and either generate a completely fake robots.txt file (if one does not exist), or modify the existing version by injecting a fake directory as a disallow directive. The "Malicious Spider Activity" incident is triggered whenever a user attempts to request a resource in the fake disallow directory, or attempts to perform a directory index on the disallow directory.

Behavior: Requesting robots.txt occurs in two different scenarios. The first is where a legitimate spider, such as Google, attempts to index the website. In this case, the robots.txt file will be requested, and no requests from that client will be issued to the disallow directories. In the second scenario, a malicious user requests robots.txt and then indexes some or all of the disallow directories. In this specific case, the user has requested robots.txt to obtain the list of disallow directories, and then started searching for resources in those directories. This activity is performed to find a "Predictable Resource Location" vulnerability. Because spidering a directory tends to be a noisy process (lots of requests), there are likely to be many of these incidents if there are any. The sum of occurrences of this incident represent the type of activity the user is performing to index a directory. The set of URL's for which this incident is triggered, represent the filenames the malicious user is testing for. For example, if they were searching for PDF files that contain stock information, there would be an incident for each filename with a PDF extension they tried to request. There is a very strong chance that if the filename was requested in the disallow directory, it was probably requested in every other directory on the site as well. This type of behavior is generally observed while the client is attempting to establish the overall attack surface of the website (or in the case of a legitimate spider, they are attempting to establish the desired index limitations).

CHAPTER 30

Activity Processors

- [Activity Processors on page 182](#)
- [Activity Processors: Custom Authentication Processor: Incident - Auth Input Parameter Tampering on page 183](#)
- [Activity Processors: Custom Authentication Processor: Incident - Auth Query Parameter Tampering on page 184](#)
- [Activity Processors: Custom Authentication Processor: Incident - Auth Cookie Tampering on page 184](#)
- [Activity Processors: Custom Authentication Processor: Incident - Authentication Brute Force on page 185](#)
- [Activity Processors: Custom Authentication Processor: Incident - Auth Invalid Login on page 185](#)
- [Activity Processors: Cookie Protection Processor on page 186](#)
- [Activity Processors: Cookie Protection Processor: Incident - Application Cookie Manipulation on page 187](#)
- [Activity Processors: Error Processor on page 187](#)
- [Activity Processors: Error Processor: Incident - Illegal Response Status on page 192](#)
- [Activity Processors: Error Processor: Incident - Suspicious Response Status on page 193](#)
- [Activity Processors: Error Processor: Incident - Unexpected Response Status on page 193](#)
- [Activity Processors: Error Processor: Incident - Unknown Common Directory Requested on page 194](#)
- [Activity Processors: Error Processor: Incident - Unknown User Directory Requested on page 194](#)
- [Activity Processors: Error Processor: Incident - Common Directory Enumeration on page 195](#)
- [Activity Processors: Error Processor: Incident - User Directory Enumeration on page 195](#)
- [Activity Processors: Error Processor: Incident - Resource Enumeration on page 196](#)
- [Activity Processors: Header Processor on page 197](#)
- [Activity Processors: Header Processor: Incident - Duplicate Request Header on page 198](#)
- [Activity Processors: Header Processor: Incident - Duplicate Response Header on page 199](#)
- [Activity Processors: Header Processor: Incident - Illegal Request Header on page 199](#)

- [Activity Processors: Header Processor: Incident - Illegal Response Header on page 200](#)
- [Activity Processors: Header Processor: Incident - Missing All Headers on page 200](#)
- [Activity Processors: Header Processor: Incident - Missing Host Header on page 200](#)
- [Activity Processors: Header Processor: Incident - Missing Request Header on page 201](#)
- [Activity Processors: Header Processor: Incident - Missing Response Header on page 201](#)
- [Activity Processors: Header Processor: Incident - Missing User Agent Header on page 202](#)
- [Activity Processors: Header Processor: Incident - Request Header Overflow on page 202](#)
- [Activity Processors: Header Processor: Incident - Unexpected Request Header on page 203](#)
- [Activity Processors: Method Processor on page 203](#)
- [Activity Processors: Method Processor: Incident - Illegal Method Requested on page 204](#)
- [Activity Processors: Method Processor: Incident - Unexpected Method Requested on page 205](#)
- [Activity Processors: Method Processor: Incident - Missing HTTP Protocol on page 205](#)
- [Activity Processors: Method Processor: Incident - Unknown HTTP Protocol on page 206](#)

Activity Processors

The custom authentication processor is designed to add strong and secure authentication to any page in the protected application. The authentication processor also logs malicious activity like invalid logins and modifying cookies or query parameters.

Table 20: Custom Authentication Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
User Accounts	Collection	[collection:0]	The list of valid user accounts to use for this processor.
Advanced			
Auth Cookie Name	String	Random	The name of the authentication cookie.
Login Page Timeout	Integer	10 Minutes	The number of seconds a login page can be used before it times out. This is intended to prevent attacks based on watching network traffic. It should be as short as is tolerable.
MD5 Script Name	String	Random	The name of the Javascript resource that contains the MD5 code.
Session Timeout	Integer	1 Hour	The number of seconds a session can be idle before it times out.

Table 20: Custom Authentication Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Incident: Auth Cookie Tampering	Boolean	True	The user has modified the cookie used to manage custom authentication, probably in an attempt to expose sensitive information or bypass access restrictions.
Incident: Auth Input Parameter Tampering	Boolean	True	The user has modified the parameters used to manage custom authentication, probably in an attempt to expose sensitive information or bypass the authentication mechanism.
Incident: Auth Invalid Login	Boolean	True	The user has attempted to login but supplied invalid credentials, this could be perfectly normal, but large numbers of this type of incident would indicate a brute force attack.
Incident: Auth Query Parameter Tampering	Boolean	True	The user has modified the query parameters that were submitted when the user was asked to originally login. This is likely in an attempt to probe the authentication mechanism for exploits.

Activity Processors: Custom Authentication Processor: Incident - Auth Input Parameter Tampering

Complexity: Medium (3.0)

Default Response: 3x = Warn User, 5x = Captcha. 9x = 1 day Clear Inputs.

Cause: WebApp Secure provides the capability of password protecting any URL on the protected site. This means that if a user attempts to access that URL, they will be prompted to enter a username and password before the original request is allowed to be completed. This incident is triggered when a user attempts to manipulate the hidden form parameters used to handle authentication.

Behavior: Manipulating hidden input fields in a form, for whatever reason is generally considered malicious. In this case, since the form is being used to password protect a resource, it is likely that the attacker is trying to bypass the authentication by finding a vulnerability in the authentication mechanism. Depending on the modified value they submit, they could be attempting to launch a "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", or "SQL injection" attack among many others.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Activity Processors: Custom Authentication Processor: Incident - Auth Query Parameter Tampering

Complexity: Low (2.0)

Default Response: 1x = Warn User. 2x = 1 day Clear Inputs.

Cause: WebApp Secure provides the capability of password protecting any URL on the protected site. This means that if a user attempts to access that URL, they will be prompted to enter a username and password before the original request is allowed to be completed. This incident is triggered when a user attempts to manipulate the query parameters that were submitted with the original unauthenticated request, after authentication has been completed.

Behavior: Manipulating query parameters after authenticating is not very easy to do without a 3rd party tool, and has no legitimate purpose. As such, this type of behavior is most likely related to a user who is trying to smuggle a malicious payload through a network or web firewall. Depending on the value the user submits for the modified query string, they could be attempting a "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", or "SQL injection" attack among many others. One interesting note is that the user has actually authenticated in order to cause this incident. As such, it is also likely that the account for which the user authenticated has been compromised and should be updated (with a new password). Although it is possible that the true owner of the account has executed the malicious action, and should therefore potentially be banned.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Activity Processors: Custom Authentication Processor: Incident - Auth Cookie Tampering

Complexity: Medium (3.0)

Default Response: 1x = Warn User, 2x = Captcha. 3x = 1 day Strip Inputs.

Cause: WebApp Secure provides the capability of password protecting any URL on the protected site. This means that if a user attempts to access that URL, they will be prompted to enter a username and password before the original request is allowed to be completed. This incident is triggered when a user attempts to manipulate the cookie used to maintain the authenticated session once the user logs in.

Behavior: Manipulating cookies is not easy to do without a 3rd party tool, and has no legitimate purpose. As such, this type of behavior is most likely related to a user who is trying to perform a "Credential/Session Prediction" attack, or execute an input based attack such as a "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", or "SQL injection" attack among

many others. One interesting note is that the user has actually authenticated in order to cause this incident. As such, it is also likely that the account for which the user authenticated has been compromised and should be updated (with a new password). Although it is possible that the true owner of the account has executed the malicious action, and should therefore potentially be banned.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Activity Processors: Custom Authentication Processor: Incident - Authentication Brute Force

Complexity: Medium (3.0)

Default Response: 1x = Captcha. 2x = 1 day Block.

Cause: WebApp Secure provides the capability of password protecting any URL on the protected site. This means that if a user attempts to access that URL, they will be prompted to enter a username and password before the original request is allowed to be completed. This incident is triggered when a user submits a large volume of invalid username and password combinations.

Behavior: Submitting a single invalid username or password is likely a user typo, and is not necessarily malicious. However it does represent a security event, and a large number of these events may represent a more serious threat such as "Brute Force". It is possible however, that the invalid username or password might also be an attack vector targeted at the authentication mechanism such as a "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", or "SQL injection" attack among many others. This incident is a higher level incident that gets tripped when dozens of "Auth Invalid Login" incidents are created. As such, it does not contain much information about the actual accounts being targeted. If more detail is desired, the underlying "Auth Invalid Login" incidents should be reviewed. These incidents are only suspicious (not considered malicious on their own), so the filtering option will need to be set to show non malicious incidents.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Activity Processors: Custom Authentication Processor: Incident - Auth Invalid Login

Complexity: Suspicious (1.0)

Default Response: 20x = Authentication Brute Force Incident.

Cause: WebApp Secure provides the capability of password protecting any URL on the protected site. This means that if a user attempts to access that URL, they will be prompted to enter a username and password before the original request is allowed to be completed. This incident is triggered when a user submits an invalid username or password. This incident alone is not necessarily malicious, as it is possible for a legitimate user to accidentally type their username or password incorrectly.

Behavior: Submitting a single invalid username or password is likely a user typo, and is not necessarily malicious. However it does represent a security event, and a large number of these events may represent a more serious threat such as "Brute Force". It is possible however, that the invalid username or password might also be an attack vector targeted at the authentication mechanism such as a "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", or "SQL injection" attack among many others. So if the value specified for the username and password does not look like a legitimate username and password (they are too long, or contain unusual characters), then this incident may be more serious. However, even in this case, the user is more likely to submit dozens of invalid credentials (not just one), and there is a different incident for that scenario.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Activity Processors: Cookie Protection Processor

This processor is responsible for protecting a set of application cookies from modification or assignment by the user.

Table 21: Cookie Protection Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Protected Cookies	Collection	Collection	The name of the protected cookie.
Advanced			
Protected Cookie Signature Suffix	String	Random	The suffix to add to the protected cookie names when generating a signature cookie. For example, if the protected cookie is PHPSESSID and the suffix is _MX, then the signature for PHPSESSID would be in a cookie named PHPSESSID_MX.
Incident: Application Cookie Manipulation	Boolean	True	The user either attempted to modify one of the protected cookies, or attempted to assign a new value.

Activity Processors: Cookie Protection Processor: Incident - Application Cookie Manipulation

Complexity: Low (2.0)

Default Response: 1x = Warn User and Logout User. 2x = 5 day Clear Inputs.

Cause: WebApp Secure is designed to provide additional protection to cookies used by the web application for tracking user sessions. This is done by issuing a signature cookie any time the web application issues a "protected cookie" (which cookies to protect is defined in configuration). The signature cookie ties the application cookie (such as PHPSESSID) to the WebApp Secure session cookie. If any of the 3 cookies are modified (WebApp Secure session cookie, signature cookie, or the actual application cookie), then this incident will be triggered, and the application cookie will be terminated (effectively terminating the users session). This prevents any users from manually creating a session cookie, hijacking another users cookie, or manipulating an existing cookie.

Behavior: Manipulation of cookies is generally performed in order to hijack another user's session. However because cookies represent another type of application input, modifications could also be performed to attempt other exploits. If the modified value resembles a legitimate value for the application cookie, then this is likely a session hijacking attempt. If the cookie contains other values that are clearly not valid, then it is more then likely an attack on generic application inputs such as a "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", and "SQL injection" attack among many others.



NOTE: For information on the attack types mentioned here, go to The [Web Application Security Consortium](#) Web Site and search for the attack name to learn more about it.

Activity Processors: Error Processor

Errors and their contents play a big part in hacking a website. When a hacker obtains an error message, it provides useful information, the very least of which is that the attacker found a way to do something unintended in the web application and the server executed code to handle it. As such, when a user attempts to hack a website, they frequently induce and receive error messages. Often these error messages are very unusual and are not common when a normal user visits the site. For example, the error code 400 (Bad Request) is returned when the raw data in a request does not follow the HTTP standards. While it is possible to get a 400 error by typing invalid characters into the URL, the majority of these errors are caused by 3rd party software (usually not a browser), improperly communicating with the server. A hacker might for example, manually construct a malicious request and forget to include the "Host" header. The goal of this processor is to record unusual and unexpected errors as incidents. This processor will also monitor all 404 errors and attempt to identify Common Directory Enumeration and User Directory Enumeration.

Table 22: Error Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Legitimate Error Detection Enabled	Boolean	True	Whether to attempt to identify errors in the protected web applications so that they can be ignored.
Advanced			
Error Cache Expiration	Integer	43200 (12 hours)	The number of seconds to cache an error condition so that subsequent matching error conditions from other users can be identified. The less traffic the site sees on a regular basis, the higher this value must be. The recommended default is for sites that see several thousand users a day or more.
Error Cache Size	Integer	50	The number of error conditions to cache for each level of specificity. If too many error conditions are encountered in a short period of time, this will prevent the tracking code from consuming too much memory. Errors at the full URL with query string specificity will cache this many conditions, at the URL only level it will cache twice this many, and at the filename level, it will cache 3 times as many as this value.
Filename Only Expiration	Integer	259200 (3 days)	The number of seconds that an error must not be encountered on a filename regardless of its location before an ignored error starts being recorded again.

Table 22: Error Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Filename Only Threshold	Integer	70	The maximum number of unique users who can hit a specific filename, regardless of location, and get the same error before it stops being recorded as suspicious (zero = do not track based on filename).
URL With Query Expiration	Integer	259200 (3 days)	The number of seconds that an error must not be encountered on the full URL with query string before an ignored error starts being recorded again.
URL With Query Threshold	Integer	30	The maximum number of unique users who can hit a full URL including query string and get the same error before it stops being recorded as suspicious (zero = do not track based on full url).
URL Without Query Expiration	Integer	259200 (3 days)	The number of seconds that an error must not be encountered on the URL excluding query string before an ignored error starts being recorded again.
URL Without Query Threshold	Integer	50	The maximum number of unique users who can hit a URL excluding query string and get the same error before it stops being recorded as suspicious (zero = do not track based on url).
100 Continue	Configurable	Error Status	Continue.
101 Switching Protocols	Configurable	Error Status	Switching Protocols.
102 Processing	Configurable	Error Status	Processing.
300 Multiple Choices	Configurable	Error Status	Multiple Choices.
301 Moved Permanently	Configurable	Error Status	Moved Permanently.
302 Found	Configurable	Error Status	Found.
303 See Other	Configurable	Error Status	See Other.

Table 22: Error Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
304 Not Modified	Configurable	Error Status	Not Modified
305 Use Proxy	Configurable	Error Status	Use Proxy.
306 Switch Proxy	Configurable	Error Status	Switch Proxy.
307 Temporary Redirect	Configurable	Error Status	Switch Proxy.
400 Bad Request	Configurable	Error Status	Bad Request
401 Unauthorized	Configurable	Error Status	Unauthorized.
402 Payment Required	Configurable	Error Status	Payment Required.
403 Forbidden	Configurable	Error Status	Forbidden
404 Not Found	Configurable	Error Status	Not Found
405 Method Not Allowed	Configurable	Error Status	Not allowed.
406 Not Acceptable	Configurable	Error Status	Not acceptable.
407 Proxy Authentication Required	Configurable	Error Status	Proxy Authentication Required
408 Request Timeout	Configurable	Error Status	Request Timeout.
409 Conflict	Configurable	Error Status	Conflict.
410 Gone	Configurable	Error Status	Gone.
411 Length Required	Configurable	Error Status	Length Required.
412 Precondition Failed	Configurable	Error Status	Precondition Failed.
413 Request Entity Too Large	Configurable	Error Status	Request Entity Too Large.
414 Request-URI Too Long	Configurable	Error Status	Request-URI Too Long.
415 Unsupported Media Type	Configurable	Error Status	Unsupported Media Type.
416 Requested Range Not Satisfiable	Configurable	Error Status	Requested Range Not Satisfiable.
417 Expectation Failed	Configurable	Error Status	Expectation Failed.
418 I'm a teapot	Configurable	Error Status	418 I'm a teapot

Table 22: Error Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
422 Unprocessable Entity	Configurable	Error Status	Unprocessable Entity.
423 Locked	Configurable	Error Status	Locked.
424 Failed Dependency	Configurable	Error Status	Failed Dependency.
425 Unordered Collection	Configurable	Error Status	Unordered Collection
426 Upgrade Required	Configurable	Error Status	Upgrade Required
449 Retry With	Configurable	Error Status	Retry With
450 Blocked by Windows Parental Controls	Configurable	Error Status	Blocked by Windows Parental Controls.
500 Internal Server Error	Configurable	Error Status	Internal Server Error
501 Not Implemented	Configurable	Error Status	Not Implemented
502 Bad Gateway	Configurable	Error Status	Bad Gateway
503 Service Unavailable	Configurable	Error Status	Service Unavailable
504 Gateway Timeout	Configurable	Error Status	Gateway Timeout
505 HTTP Version Not Supported	Configurable	Error Status	HTTP Version Not Supported
506 Variant Also Negotiates	Configurable	Error Status	Variant Also Negotiates
507 Insufficient Storage	Configurable	Error Status	Insufficient Storage
509 Bandwidth Limit Exceeded	Configurable	Error Status	Bandwidth Limit Exceeded
510 Not Extended	Configurable	Error Status	Not Extended
Incident: Illegal Response Status	Boolean	True	The user issued a request that resulted in an error status code that is considered suspicious and possibly malicious.

Table 22: Error Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Incident: Suspicious Response Status	Boolean	True	The user issued a request that resulted in a known error status code generally involved in malicious behavior. On its own this is not enough to classify abuse, but patterns of this indicator may lead to higher level malicious incidents.
Incident: Unexpected Response Status	Boolean	True	The user issued a request that resulted in an unknown error status code and could represent a successful exploit.
Incident: Unknown Common Directory Requested	Boolean	True	The user has requested a directory that does not exist. The directory is in a list of common directory names, so it is likely that this request is in an attempt to find a directory that is not linked from the site.
Incident: Unknown User Directory Requested	Boolean	True	The user has requested a directory for a specific system user that does not exist. The username is in a list of common usernames, so it is likely that this request is in an attempt to identify a user account that is not linked from the site.

Activity Processors: Error Processor: Incident - Illegal Response Status

Complexity: Suspicious (1.0)

Default Response: None.

Cause: WebApp Secure monitors the various status codes returned by the protected website and compares them to a configurable list of known and acceptable status codes. Some status codes are expected during normal usage of the site (such as 200 - OK, or 403 - Not Modified), but some status codes are much less common for a normal user (such as 500 - Server Error, or 404 - File Not Found). When a user issues a request that results in a status code that is configured as illegal, then this incident will be triggered.

Behavior: In the process of attempting to find vulnerabilities on a web server, hackers will often encounter errors. Just a single error or two is likely not a problem, because even legitimate users accidentally type a URL incorrectly on occasion. However when excessive numbers of unexpected status codes are returned, the behavior of the user can be

narrowed down and classified as malicious. The actual vulnerability an attacker is looking for, can be identified through the status codes they are being returned. For example, if the user is getting a lot of 404 errors, they are likely searching for unlinked files ("Predictable Resource Location"). If the user is getting a lot of 500 errors, they may be trying to establish a successful "SQL Injection" or "XSS150" vulnerability.

Activity Processors: Error Processor: Incident - Suspicious Response Status

Complexity: Suspicious (1.0)

Default Response: 10x 404 = Resource Enumeration Incident.

Cause: WebApp Secure monitors the various status codes returned by the protected website and compares them to a configurable list of known and acceptable status codes. Some status codes are expected during normal usage of the site (such as 200 - OK, or 403 - Not Modified), but some status codes are much less common for a normal user (such as 500 - Server Error, or 404 - File Not Found). When a user issues a request which results in a status code that is not known and does not have any associated configuration, this incident will be triggered.

Behavior: In the process of attempting to find vulnerabilities on a web server, hackers will often encounter errors. Just a single error or two is likely not a problem, because even legitimate users accidentally type a URL incorrectly on occasion. However when excessive numbers of unexpected status codes are returned, the behavior of the user can be narrowed down and classified as malicious. The actual vulnerability the attacker is looking for can be identified through the status codes they are being returned. For example, if the user is getting a lot of 404 errors, they are likely searching for unlinked files ("Predictable Resource Location"). If the user is getting a lot of 500 errors, they may be trying to establish a successful "SQL Injection" or "XSS" vulnerability. In the case of this incident, the user is getting an unexpected status code. This is likely because of a bug in the web application which the user has found and is attempting to exploit. The URL this incident is created for, should be reviewed to determine why it would be responding with a non standard status code. If the status code is intentionally non-standard, but is acceptable behavior, then the custom status code should be added to the list of known and accepted status codes in config.

Activity Processors: Error Processor: Incident - Unexpected Response Status

Complexity: Suspicious (1.0)

Default Response: None.

Cause: WebApp Secure monitors the various status codes returned by the protected website and compares them to a configurable list of known and acceptable status codes. Some status codes are expected during normal usage of the site (such as "200 OK" or "304 Not Modified"), but some status codes are much less common for a normal user (such as "500 Internal Server Error" or "404 Not Found"). When a user issues a request which results in a status code that is not known and does not have any associated configuration, this incident will be triggered.

Behavior: In the process of attempting to find vulnerabilities on a web server, hackers will often encounter errors. Just a single error or two is likely not a problem, because even legitimate users accidentally type a URL incorrectly on occasion. However when excessive numbers of unexpected status codes are returned, the behavior of the user can be narrowed down and classified as malicious. The actual vulnerability the attacker is looking for can be identified through the status codes they are being returned. For example, if the user is getting a lot of 404 errors, they are likely searching for unlinked files ("Predictable Resource Location"). If the user is getting a lot of 500 errors, they may be trying to establish a successful "SQL Injection" or "XSS" vulnerability. In the case of this incident, the user is getting an unexpected status code. This is likely because of a bug in the web application which the user has found and is attempting to exploit. The URL this incident is created for, should be reviewed to determine why it would be responding with a non standard status code. If the status code is intentionally non-standard, but is acceptable behavior, then the custom status code should be added to the list of known and accepted status codes in config.

Activity Processors: Error Processor: Incident - Unknown Common Directory Requested

Complexity: Suspicious (1.0)

Default Response: 5x = Common Directory Enumeration Incident

Cause: This incident is triggered when a user requests a directory on the server that does not exist, and that directory name is in a list of commonly used directory names (for example: <http://www.example.com/public/> where "public" is not a real directory).

Behavior: Often times, administrators will upload sensitive content onto a web server in an obscure location and not link to that content anywhere on the site. The assumption is that the content is private because no one will find it. However humans are somewhat predictable, so it's actually quite common for two administrators to pick the same "obscure" location to place sensitive content. As such, hackers have compiled a list of the most commonly chosen directory names where sensitive content is often stored, and they will basically test every name in the list to see if a site has a directory by that name. If it does, the attacker is able to locate and obtain that sensitive content. An example of a tool that allows attackers to quickly identify hidden directories is called "DirBuster" (https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project).

Activity Processors: Error Processor: Incident - Unknown User Directory Requested

Complexity: Suspicious (1.0)

Default Response: 5x = User Directory Enumeration Incident

Cause: Many web servers allow the users on the system to maintain publicly accessible web directories. These directories are generally accessible from the root directory of the website followed by a tilde and the username. For example, if the web server had a user named 'george', that user could serve content from <http://www.example.com/~george/>. This incident is triggered when an attacker requests a user directory on the server that does not exist, and that user directory name is in a list of commonly used usernames (for example: <http://www.example.com/~root/> where "root" is not a real user directory).

Behavior: Often times, administrators will upload sensitive content onto a web server in an obscure location and not link to that content anywhere on the site. The assumption is that the content is private because no one will find it. However humans are somewhat predictable, so it's actually quite common for two administrators to pick the same "obscure" location to place sensitive content. As such, hackers have compiled a list of the most commonly chosen directory names where sensitive content is often stored, and they will basically test every name in the list to see if a site has a directory by that name. If it does, the attacker is able to locate and obtain that sensitive content. In this specific case, the attacker is testing for default user directories for users with predictable names (such as 'root', 'guest', 'nobody', etc...). An example of a tool that allows attackers to quickly identify hidden user directories is called "DirBuster" (https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project).

Activity Processors: Error Processor: Incident - Common Directory Enumeration

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection 2-6 seconds & Captcha, 2x = Slow Connection 2-6 seconds & 1 day Block

Cause: This incident is triggered when a user requests a directory on the server that does not exist, and that directory name is in a list of commonly used directory names (for example: <http://www.example.com/public/> where "public" is not a real directory). Specifically, this incident is triggered when the user requests many different commonly named directories, as would be the case if they were testing for a large list of possible directory names.

Behavior: Often times, administrators will upload sensitive content onto a web server in an obscure location and not link to that content anywhere on the site. The assumption is that the content is private because no one will find it. However humans are somewhat predictable, so it's actually quite common for two administrators to pick the same "obscure" location to place sensitive content. As such, hackers have compiled a list of the most commonly chosen directory names where sensitive content is often stored, and they will basically test every name in the list to see if a site has a directory by that name. If it does, the attacker is able to locate and obtain that sensitive content. An example of a tool that allows attackers to quickly identify hidden directories is called "DirBuster" (https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project).

Activity Processors: Error Processor: Incident - User Directory Enumeration

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection 2-6 seconds & Captcha, 2x = Slow Connection 2-6 seconds & 1 day Block

Cause: Many web servers allow the users on the system to maintain publically accessible web directories. These directories are generally accessible from the root directory of the website followed by a tilde and the username. For example, if the web server had a user named 'george', that user could serve content from <http://www.example.com/~george/>. This incident is triggered when an attacker requests a user directory on the server that

does not exist, and that user directory name is in a list of commonly used usernames (for example: `http://www.example.com/~root/` where "root" is not a real user directory). Specifically, this incident is triggered when an attacker requests many different username directories, as would be the case if they were testing for a large list of possible usernames.

Behavior: Often times, administrators will upload sensitive content onto a web server in an obscure location and not link to that content anywhere on the site. The assumption is that the content is private because no one will find it. However humans are somewhat predictable, so it's actually quite common for two administrators to pick the same "obscure" location to place sensitive content. As such, hackers have compiled a list of the most commonly chosen directory names where sensitive content is often stored, and they will basically test every name in the list to see if a site has a directory by that name. If it does, the attacker is able to locate and obtain that sensitive content. In this specific case, the attacker is testing for default user directories for users with predictable names (such as 'root', 'guest', 'nobody', etc). An example of a tool that allows attackers to quickly identify hidden user directories is called "DirBuster"

(https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project).

Activity Processors: Error Processor: Incident - Resource Enumeration

Complexity: Low (2.0)

Default Response: 1x = 5 day Block.

Cause: WebApp Secure has a list of file tokens which represent potentially sensitive files. For example, developers will often rename source files with a ".bck" extension during debugging, and sometimes they forget to delete the backup after they are done. Hackers often look for these left over source files. WebApp Secure is configured to look for any request to a file with a ".bck" extension (as well as any other configured extensions), and trigger a Suspicious Filename incident if the file does not exist. Should the suspicious filename incident be triggered several times, this incident will then be triggered.

Behavior: There are specific files that many web sites host, that contain valuable information for a hacker. These files generally include data such as passwords, SQL schema's, source code, etc. When hackers try to breach a site, they will often check to see if they can locate some of these special files in order to make their jobs easier. For example, if a hacker sees that the home page is called "index.php", they may try and request "index.php.bak", because if it exists, it will be returned as raw source code. This is usually an effort to exploit a "Predictable Resource Location68" vulnerability. Automated scanners will generally test all of these types of extensions (.bck, .bak, .zip, .tar, .gz, etc...) against every legitimate file that is located through simple spidering. The first few times a user requests a filename containing a suspicious token, they will only get "Suspicious Filename" incidents. However if they request a large volume of filenames with suspicious tokens, then the "Suspicious Resource Enumeration" incident is generated. This incident represents a user who is actively scanning the site with very aggressive tactics to find unlinked and sensitive data.

Activity Processors: Header Processor

A useful technique when attacking a site is to determine what software the site is using. This is known as fingerprinting the server. There are many methods used, but the basic idea is to look for signatures that identify various products. For example, it might be a known signature that Apache always lists the **Date** response header before the **Last-Modified** response header. If very few other servers follow this same pattern, then checking to see which header comes first could be used as a means of identifying if Apache is being used or not. Other key methods include looking for **Server** or **X-Powered-By** headers that actually specify the software being used. The goal of this processor is to eliminate headers as a means of fingerprinting a server.



NOTE: While the goal of this processor is mainly to prevent fingerprinting, it may also catch some malicious behavior and erroneous behavior in the protected applications (potentially as a result of an exploit). As such, the following incidents are recognized by the processor.

Table 23: Header Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Header Mixing Enabled	Boolean	True	Whether this processor should shuffle the order of response headers to avoid exposing identifiable information.
Request Header Stripping Enabled	Boolean	True	Whether this processor should strip unnecessary headers in request packets to avoid sending malicious data to the server.
Response Header Stripping Enabled	Boolean	True	Whether this processor should strip unnecessary response headers to avoid giving away identifiable information.
Maximum Header Length	Integer	8192	The maximum allowed length of a header in bytes. If header stripping is enabled, then any headers that exceed this length will be removed from the request before proxying.
Known Request Headers	Collection	Collection	A list of known request headers.
Known Response Headers	Collection	Collection	A list of known Response headers.

Table 23: Header Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Incident: Duplicate Request Header	Boolean	True	The application returned multiple instances of the same header, which it is never expected to do.
Incident: Duplicate Response Header	Boolean	False	The user provided multiple instances of the same header, and the header does not usually allow multiples.
Incident: Illegal Request Header	Boolean	True	The user provided a request header which is known to be involved in malicious activity.
Incident: Illegal Response Header	Boolean	True	The application returned a response header which it is never supposed to return.
Incident: Missing All Headers	Boolean	True	The user issued a request which has no headers at all.
Incident: Missing Host Header	Boolean	True	The application returned a response which is missing a required header.
Incident: Missing Request Header	Boolean	True	The user issued a request which is missing a required header.
Incident: Missing Response Header	Boolean	True	The application returned a response which is missing a required header.
Incident: Missing User Agent Header	Boolean	True	The user issued a request which is missing a required header.
Incident: Request Header Overflow	Boolean	True	The user issued a request which contained a header that was longer than the allowed maximum.
Incident: Unexpected Request Header	Boolean	False	The user issued a request which contains an unexpected and unknown header.

Activity Processors: Header Processor: Incident - Duplicate Request Header

Complexity: Informational (0.0)

Default Response: None

Cause: WebApp Secure monitors all of the request headers sent from the client to the web application. According to the HTTP RFC, no client should ever provide more than one copy of a specific header. For example, clients should not send multiple Host headers. However there are a few exceptions, such as the Cookie header, which can be configured to allow multiples. If the user sends multiple headers that are not configured explicitly to allow duplicates, then this incident will be triggered.

Behavior: Sending duplicate headers of the same type can be caused by several different things. It is either an attempt to profile the web server and see how it reacts, an attempt

to smuggle malicious data into the headers (because a firewall might not look at subsequent copies of the same header), or possibly just be a poorly programmed web client. In either case, it represents unusual activity that sets the user aside from everyone else. It signifies that the user is suspicious and is doing something average users do not do.

Activity Processors: Header Processor: Incident - Duplicate Response Header

Complexity: Informational (0.0)

Default Response: None

Cause: Secure monitors all of the response headers sent from the server to the client. According to the HTTP RFC, no server should ever provide more than one copy of a specific header. For example, servers should not send multiple "Content-Length" headers. However, there are a few exceptions, such as the "Set-Cookie" header, which can be configured to allow multiples. If the server attempts to return multiple headers of the same type, which are not configured explicitly to allow duplicates, then this incident will be triggered.

Behavior: The RFC does not allow for servers to return multiple headers of the same type, with a few exceptions, such as Set-Cookie. If the server does return duplicates for a header that normally does not support duplicates, then there is either a bug in the web application, or the user has successfully executed a "Response Splitting" attack. In either case, the service located at the URL this incident is triggered for should probably be reviewed for response splitting vulnerabilities or bugs that would cause duplicate response headers to be returned.

Activity Processors: Header Processor: Incident - Illegal Request Header

Complexity: Suspicious (1.0)

Default Response: None.

Cause: WebApp Secure monitors all of the request headers included by clients. It has a list of known request headers that should never be accepted. This list is configurable, and by default, includes any headers known to be exclusively involved in malicious activity. Should a user include one of the illegal headers, this incident will be triggered. Because the list of illegal headers is configurable, it cannot be guaranteed that the request that contained the header is strictly malicious, but it does signify that the client is doing something highly unusual.

Behavior: Some HTTP headers can be used in order to get the server to do something it isn't designed to do. For example, the "max-forwards" header can be used to specify how many hops within the internal network the request should make before it is dropped. An attacker could use this header to identify how many network devices are between themselves and the target web server. Because the list of illegal headers is customizable, the type of behavior the header relates to can vary. However, this type of behavior is generally performed when scoping the attack surface of the web site.

Activity Processors: Header Processor: Incident - Illegal Response Header

Complexity: Informational (0.0)

Default Response: None.

Cause: WebApp Secure monitors all of the response headers sent to the client from the web application. It has a list of known response headers that should never be returned. This list is configurable, and by default, includes any headers known to compromise the server's identity or security. Should the server return one of the illegal headers, this incident will be triggered. Because the list of illegal headers is configurable, it cannot be guaranteed that the request that contained the header is strictly malicious, but it does signify that something unusual has taken place. This may even represent a hackers successful attempt to exploit a backend service.

Behavior: There is a strict set of HTTP response headers that browsers understand and can actually use. Any headers returned by the server outside of the standard set could potentially expose information about the server or its software. Some headers can even be used to execute more complex attacks. In order to protect the server in the event of a serious issue (such as a "Response Splitting¹⁵⁹" attack), some headers can be configured as illegal. Because the set is configurable, it is not straight forward as to what the actual header means or what vulnerability it might be targeted at.

Activity Processors: Header Processor: Incident - Missing All Headers

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds and Captcha.

Cause: Most legitimate web browsers and tools submit at least a few headers with each HTTP request. Headers are used to provide valuable information to the server when trying to construct a response, such as what type of browser the user is using, or what domain name they are trying to access. If a user submits a request that does not contain any headers at all, this incident will be triggered.

Behavior: Not providing any headers at all, is generally an activity performed when probing an IP to see if it is running a web server. The user will submit a minimal request containing 1 line of text, and see if the response given back from the server is an HTTP response. If so, the attacker has confirmed that the IP is hosting a web server on the given port. In many cases, the attacker will also be able to identify which web server is running, and if that web server has any known vulnerabilities. Such information can then be used to attack the web server directly.

Activity Processors: Header Processor: Incident - Missing Host Header

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds and Captcha.

Cause: All legitimate web browsers submit a Host header with each HTTP request. The host header contains the value entered into the address bar as the server. This could be either the server IP address or the domain name. In either case, it will always be provided. If a user submits a request that does not contain a Host header, this incident will be triggered.

Behavior: Not providing a host header is generally an activity performed when trying to scope the attack surface of the web site. Some web servers are configured to host different web sites from the same IP address, based on which domain name is supplied. Hackers will often attempt to send a request without a host header to see if the server will serve back a default web site. If the default web site is not the main web site, this may provide additional pages the attacker can attempt to exploit. This could be considered a "Server Misconfiguration" weakness, but may also be a legitimate design choice for the web server and its applications. It does not necessarily expose a vulnerability as long as the default web application is secure. Because all major browsers submit host headers on every request, the user would need to take advantage of a more complex tool, such as a raw data client, or HTTP debugging proxy to manually construct a request that does not have a host header. As such, this activity is almost always malicious. In a few cases, some legitimate monitoring tools may omit this header, but those tools should be added to the trusted IP list in configuration.

Activity Processors: Header Processor: Incident - Missing Request Header

Complexity: Low (2.0)

Default Response: None.

Cause: WebApp Secure monitors all of the request headers sent from the client to the server. It also maintains a list of headers which are required for all HTTP requests (such as Host and User-Agent). If one of the required headers is not included in a request, this incident will be triggered.

Behavior: Every legitimate client will always supply specific headers such as "Host" and "User-Agent". If a client does not provide these headers, then the client is likely not a legitimate user. There are several different cases of not legitimate clients, such as hacking tools, manually crafted HTTP requests using something like Putty, or a network diagnostic tool such as nagios. Because there are a few cases that are not necessarily malicious (such as nagios), the incident itself is not necessarily malicious. It does however exclude the user from being a legitimate web browser doing the intended actions allowed by the web application.

Activity Processors: Header Processor: Incident - Missing Response Header

Complexity: Informational (0.0)

Default Response: None.

Cause: WebApp Secure monitors all of the response headers sent from the server to the client. It also maintains a list of headers which are required for all HTTP responses (such as Content-Type). If one of the required headers is not included in a response, this incident will be triggered.

Behavior: If the server is acting correctly, it should always return all of the required response headers. If it is missing a response header, this is likely due to a bug in the web application, or a successfully executed "Response Splitting" attack. In either case, the service located at the URL this incident is triggered for, should probably be reviewed for either response splitting vulnerabilities, or bugs that would cause abnormal HTTP responses (such as dropping the connection immediately after sending the status code).

Activity Processors: Header Processor: Incident - Missing User Agent Header

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds and Captcha.

Cause: Most legitimate web browsers and tools submit a User-Agent header with each HTTP request. The user agent header contains information that identifies which software the user is using to access the website, whether that software it is Googlebot, Firefox, Safari, or another piece of software. If a user submits a request that does not contain a User-Agent header, this incident will be triggered.

Behavior: Not providing a user-agent header is generally an activity performed trying to evade detection. The user agent header provides identifying information that could be used by the web server to track requests made by the same user. It may also provide information about the user's personal computer. Sometimes, hackers will replace the user agent string with another user agent string that is perfectly legitimate, but for a different environment than the one they are actually using. Some legitimate users also take this measure as a general security practice; therefore, as long as at least some value is submitted for the user-agent, it cannot be guaranteed to be a malicious act. However, in the case of the header being absent, a user would have had to take advantage of a tool or debugging proxy in order to filter the traffic. This is almost always performed during the course of a malicious action. Some tools such as network health monitors may also trigger this incident, because they are doing something normal users should not do, but they are considered trusted. In this case, the IP addresses of those tools should be added to the configuration trusted IP whitelist.

Activity Processors: Header Processor: Incident - Request Header Overflow

Complexity: Suspicious (1.0)

Default Response: 3x = Compound Request Header Overflow Incident.

Cause: WebApp Secure monitors all of the request headers sent from the client to the server. It has a configured limit that defines how long any individual header is allowed to be. After 3 or more headers are submitted that exceed the limit, this incident will be triggered.

Behavior: While not as common as form inputs or query parameter inputs, some web applications actually use the values submitted in headers within their code base. If these values are treated incorrectly, such as not being validated before being used in an SQL statement, they potentially expose the same set of vulnerabilities a form input might. As such a hacker who is attempting to execute a "Buffer Overflow162" attack might do so by attempting to provide an excessively long value in a header. They may also use an

excessively long header value to craft a complex "SQL Injection" attack. Because the user submitted multiple headers which exceeded the defined limit, the intentions of the user are more likely to be malicious. It is less likely that a poorly crafted browser plug-in would overflow multiple headers, despite the possibility that it might overflow a single one. Because there is a possibility that a legitimate user with a poorly-written browser plugin may cause a header of unusual length to be submitted, this incident cannot be guaranteed to be malicious from just a single case.

Activity Processors: Header Processor: Incident - Unexpected Request Header

Complexity: Informational (0.0)

Default Response: None

Cause: WebApp Secure monitors all of the request headers included by clients. It has a list of known request headers that should be accepted. This list includes all of the headers defined in the HTTP RFC document, which means that if any additional headers are passed, it is part of some non standard HTTP extension. Should a user include a non standard header, this incident will be triggered. It is not necessarily a malicious action on its own, but it does signify that the client is unusual in some way (and potentially malicious) and therefore warrants additional monitoring.

Behavior: When attackers are trying to exploit a server, one of the techniques is to attempt to profile what software the server is running. This can be partially accomplished by observing how the server reacts to various types of headers. For example, if the attacker knows that a specific 3rd party web application has a feature where it behaves differently if you send a header "X-No-Auth", then a hacker might send "X-No-Auth" to the site just to see what happens. While this could represent a higher level attack on a specific application; sending non standard headers is more likely part of the hacker's effort to scope the attack surface of the web site. This incident alone cannot be deemed malicious because some users have browser plug-ins installed that automatically include non standard headers with requests to some sites. Additionally, some AJAX sites also pass around custom headers as part of their expected protocol.

Activity Processors: Method Processor

GET and POST are two very well known HTTP request methods. A request method is a keyword that tells the server what type of request the user is making. In the case of a GET, the user is requesting a resource. In the case of a POST, the user is submitting data to a resource. There are however, several other supported request methods which include HEAD, PUT, DELETE, TRACE, and OPTIONS. These methods are intended to divide the types of requests into more granular operation. In almost all web application implementations, the PUT, DELETE, TRACE and OPTIONS methods are all left unimplemented. Unfortunately, some systems provide default implementations for things such as TRACE and OPTIONS. As a result, some administrators accidentally expose unprotected services. Hackers often try these different request methods to identify servers which support them, and therefore may be vulnerable.

Table 24: Method Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			Whether traffic should be passed through this processor.
Processor Enabled	Boolean	True	
Advanced			
Block Unknown Methods	Boolean	True	Whether to block requests that contain unknown HTTP methods.
Block Unknown Protocol	Boolean	True	Whether to block requests that contain unknown HTTP protocols.
Known Methods	Collection	Collection	The list of known HTTP methods. Also allows you to customize the action to take for each occurrence of the known HTTP method.
Incident: Illegal Method Requested	Boolean	True	The user issued a request using an HTTP method which is considered illegal.
Incident: Unexpected Method Requested	Boolean	True	The user issued a request using a request method other than GET, POST, and HEAD, which resulted in a server error.
Incident: Missing HTTP Protocol	Boolean	True	No protocol specified in GET line.
Incident: Unknown HTTP Protocol	Boolean	True	Non standard protocol specified in GET line (anything except 0.9, 1.0, 1.1).

Activity Processors: Method Processor: Incident - Illegal Method Requested

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds and 1 Day Clear Inputs in 10 minutes

Cause: HTTP supports several different "methods" of submitting data to a web server. These methods generally include "GET", "POST", and "HEAD", and less commonly "PUT", "DELETE", "TRACE", and "OPTIONS". WebApp Secure monitors all of the methods used by a user when issuing HTTP requests, and compares them to a configured list of known and allowed HTTP methods. If the user submits a request that uses a method which is not in the list of known methods, this incident will be triggered.

Behavior: HTTP methods allow the web server to handle user provided data in different ways. However some of the supported methods are somewhat insecure and should not be supported unless absolutely necessary. In a few cases, methods which are not standard to HTTP are used by 3rd party web applications. When an attacker is looking for a known vulnerability, they may issue requests using some of these custom defined HTTP methods to see if the server accepts or rejects the request. If the server accepts the request, then the software is likely installed. This type of activity is generally performed when scoping the attack surface of the web application. It is possible that if a third-party web application is legitimately installed and is using custom HTTP methods, that those methods will need to be added to the list of configured HTTP methods so as not to flag users who are using those applications. In either case, because it is possible for this incident to happen without malicious intent, it is considered only suspicious.

Activity Processors: Method Processor: Incident - Unexpected Method Requested

Complexity: Suspicious (1.0)

Default Response: None.

Cause: HTTP supports several different "methods" of submitting data to a web server. These methods generally include "GET", "POST", and "HEAD", and less commonly "PUT", "DELETE", "TRACE", and "OPTIONS". WebApp Secure monitors all of the methods used by a user when issuing HTTP requests, and compares them to a configured list of known and allowed HTTP methods. If the user submits a request that uses a method which is not in the list of known methods, this incident will be triggered.

Behavior: HTTP methods allow the web server to handle user provided data in different ways. However some of the supported methods are somewhat insecure and should not be supported unless absolutely necessary. In a few cases, methods which are not standard to HTTP are used by 3rd party web applications. When an attacker is looking for a known vulnerability, they may issue requests using some of these custom defined HTTP methods to see if the server accepts or rejects the request. If the server accepts the request, then the software is likely installed. This type of activity is generally performed when scoping the attack surface of the web application. It is possible that if a 3rd party web application is legitimately installed and is using custom HTTP methods, that those methods will need to be added to the list of configured HTTP methods so as not to flag users who are using those applications. In either case, because it is possible for this incident to happen without malicious intent, it is considered only suspicious.

Activity Processors: Method Processor: Incident - Missing HTTP Protocol

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection 2-6 seconds & 1 Hour Clear Inputs

Cause: HTTP comes in several different versions. These are specified in each request issued by a client to the web server. The acceptable standard versions are 0.9, 1.0, and 1.1. Any other protocol represents a nonstandard HTTP request issued by a non-standard HTTP client. Under nearly every legitimate use-case, there is no reason to either omit the protocol or to provide one that is not standard. This incident triggers whenever a user

submits a request that is completely missing a protocol version. This would represent a clear violation of the HTTP protocol RFC specifications.

Behavior: This incident is likely to occur whenever the attacker is attempting to create a custom attack script against the web site. They may have either forgotten to include a protocol value, or they are intentionally omitting it to prevent intended functionality by one of the devices that processes the request. For example, an attacker may try to submit a request without a protocol in an effort to break security devices protecting the web server. These security devices may not be able to handle non-standard protocols correctly, and as a result, may allow malicious requests to reach the backend unmodified.

Activity Processors: Method Processor: Incident - Unknown HTTP Protocol

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection 2-6 seconds & 1 Hour Clear Inputs

Cause: HTTP comes in several different versions. These are specified in each request issued by a client to the web server. The acceptable standard versions are 0.9, 1.0, and 1.1. Any other protocol represents a nonstandard HTTP request issued by a non-standard HTTP client. Under nearly every legitimate use-case, there is no reason to either omit the protocol or to provide one that is not standard. This incident triggers whenever a user submits a request that contains an unknown protocol version. This would represent a clear violation of the HTTP protocol RFC specifications. The only time this should be acceptable behavior, is if the web application intentionally utilizes a non-standard protocol, however this should rarely, if ever, be the case.

Behavior: This incident is likely to occur whenever the attacker is attempting to create a custom attack script against the web site. They may have either mistyped the protocol value, or they are intentionally using a non-standard value to prevent intended functionality by one of the devices that processes the request. For example, an attacker may try to submit a request with an invalid protocol of 11.1 in an effort to break security devices protecting the web server. These security devices may not be able to handle non-standard protocols correctly, and as a result, may allow malicious requests to reach the backend unmodified.

CHAPTER 31

Tracking Processors

- [Tracking Processors: Etag Beacon Processor on page 207](#)
- [Tracking Processors: Etag Beacon Processor: Incident - Session Etag Spoofing on page 208](#)
- [Tracking Processors: Client Beacon Processor on page 209](#)
- [Tracking Processors: Client Beacon Processor: Incident - Beacon Parameter Tampering on page 210](#)
- [Tracking Processors: Client Beacon Processor: Incident - Beacon Session Tampering on page 211](#)
- [Tracking Processors: Client Fingerprint Processor on page 211](#)
- [Tracking Processors: Client Fingerprint Processor: Incident - Fingerprint Directory Indexing on page 214](#)
- [Tracking Processors: Client Fingerprint Processor: Incident - Fingerprint Directory Probing on page 214](#)
- [Tracking Processors: Client Fingerprint Processor: Incident - Fingerprint Manipulation on page 215](#)
- [Tracking Processors: Client Classification Processor on page 215](#)

Tracking Processors: Etag Beacon Processor

This processor is not intended to identify hacking activity, but instead is intended to help resolve a potential vulnerability in the proxy. Because session tracking in the proxy is done using cookies, it is possible for an attacker to clear their cookies in order to be recognized by the proxy as a new user. This means that if we identify that someone is a hacker, they can shed that classification simply by clearing their cookies. To help resolve this vulnerability, this processor attempts to store identifying information in the browsers JavaScript persistence mechanism. It then uses this information to attempt to identify new sessions as being created by the same user as a previous session. If successful, a hacker who clears their cookies and obtains a new session will be re-associated with the previous session shortly afterwards.

Table 25: Etag Beacon Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			

Table 25: Etag Beacon Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Beacon Resource	Configurable	Random	The resource to use for tracking.
Inject Beacon Enabled	Boolean	True	Whether a reference to the beacon resource should be automatically injected into HTML responses.
Revalidation Frequency	Integer	180 (3 Minutes)	How often in seconds to re-validate the old stored etag and re-associate that session with the current one. This value should not be left too short, because it will cause the browser to constantly re-request the fake resource and make the tracking technique more visible.
Incident: Session Etag Spoofing	Boolean	True	The user has provided a fake ETag value which is not a valid session.

Tracking Processors: Etag Beacon Processor: Incident - Session Etag Spoofing

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection 2-6 seconds, 3x = Slow Connection 4-15 seconds.

Cause: The HTTP protocol supports many different types of client side resource caching in order to increase performance. One of these caching mechanisms uses a special header called "E-Tag" to identify when the client already has a valid copy of a resource. When a user requests a resource for the first time, the server has the option of returning an E-Tag header. This header contains a key that represents the version of the file that was returned (ex. an MD5 hash of the file contents). On subsequent requests for the same resource, the client will provide the last E-Tag it was given for that resource. If the server identifies that both the provided E-Tag, and the actual E-Tag of the file are the same, then it will respond with a 403 status code (Not Modified), and the client will display the last copy it successfully downloaded. This prevents the client from downloading the same version of a resource over and over again. In the event that the E-Tag value does not match, the server will return a new copy of the resource and a new E-Tag value. WebApp Secure takes advantage of this caching mechanism to store a tracking token on the client. It does this by injecting a fake embedded resource reference (such as an image or a JavaScript file) into some of the pages on the protected site. When the browser loads these pages, it will automatically request the embedded resources in the background. The fake resource that was injected by WebApp Secure, will supply a special E-Tag value that contains a tracking token. As the user continues to navigate around the site, each time they load a page that contains a reference to the fake resource, the browser will automatically transmit the previously received E-Tag to the server. This allows WebApp Secure to correlate the requests, even if other tracking mechanisms such as cookies are not successful. The E-Tag value returned by the fake resource, which contains the tracking token, is also digitally signed and encrypted, much like the WebApp Secure

session cookie. This prevents a user from successfully guessing a valid E-Tag token, or attempting to provide an arbitrary value without being detected. If an invalid E-Tag is supplied for the fake resource, a "Session ETag Spoofing" incident is triggered.

Behavior: There are very few cases where the E-Tag caching mechanism is part of an attack vector, so this incident would almost exclusively represent a user who is attempting to evade tracking or exploit the tracking method to their advantage. For example, if a user identifies the E-Tag tracking mechanism, they may provide alternate values in order to generate errors in the tracking logic and potentially disconnect otherwise correlated traffic. They may also attempt to guess other valid values in order to correlate otherwise nonrelated traffic (such as a hacker attempting to group other legitimate users into their traffic). While this is a highly unlikely attack vector, it could loosely be classified as a "Credential and Session Prediction" attack. It is also possible, though unlikely, that once an attacker identifies the dynamic nature of the E-Tag header for the fake resource, they may also launch a series of other attacks based on input manipulation. This could include testing for SQL injection, XSS, Buffer Overflow, Integer Overflow, and HTTP Response Splitting among others. However these would be attacks directly against WebApp Secure, and not against the protected web application.

Tracking Processors: Client Beacon Processor

The client beacon processor is intended to digitally tag users for later identification by for embedding a tracking token into the client. There are configurable parameters that administrators can use to configure each type of storage mechanisms that are used track malicious users.

Table 26: Client Beacon Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Flash Storage Enabled	Boolean	True	Whether to use the flash shared data API to track the user.
IE UserData Storage Enabled	Boolean	True	Whether to use internet explorers userData storage API to track the user.
Local Storage Enabled	Boolean	True	Whether to use Javascript local storage to track the user.
Private Storage Enabled	Boolean	True	Whether to track users between private browsing mode and normal browsing mode in Firefox. A collection of names to use for the Application session cookie.

Table 26: Client Beacon Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Silverlight Storage Enabled	Boolean	True	Whether to use the Silverlight storage api to track the user. The Silverlight storage API is unique in that it is exposed across all browsers. If this beacon is enabled and the user has Silverlight installed, this beacon can track the user even if they switch browsers.
Window Name Storage Enabled	Boolean	True	Whether to use the window.name property of the browser window to track the user.
Resource Extensions	Collection	Collection	A collection of resource extensions to use for the processor.
Script Refresh Delay	Integer	3600 (1 Hour)	The amount of time in seconds to cache the randomly generated set of beacon scripts. After this amount of time, the beacon scripts will change.
Script Variations	Integer	30	The number of random variations of the beacon script to cache, and then to select from on each request.
Incident: Beacon Parameter Tampering	Boolean	True	The user has issued a request to the session tracking service which appears to be manually crafted. This is likely in an attempt to spoof another users session, or to exploit the applications session management. This would never happen under normal usage.
Incident: Beacon Session Tampering	Boolean	True	The user has altered the data stored on the client in an effort to prevent tracking. They have altered the data in such a way as to remain consistent with the same data format. This would never happen under normal usage.

Tracking Processors: Client Beacon Processor: Incident - Beacon Parameter Tampering

Complexity: Medium (3.0)

Default Response: 1x = 5 day Clear Inputs in 10 minutes

Cause: WebApp Secure uses a special persistent token that inserts itself in multiple locations throughout the client. When a user returns to the site later on, these tokens are transmitted back to the server. This allows the server to correlate the traffic issued by the same user, even if the requests are weeks apart. This incident is triggered when the user manipulates the token data being transmitted to the server on a subsequent visit. They manipulated the data in such a way as to break the expected formatting for the token.

Behavior: Attempts to manipulate and spoof the tracking tokens are generally performed when the attacker is trying to figure out what the token is used for and potentially evade tracking. Because the format of the token is completely wrong, this is likely a generic input attack, where the user is attempting to find a vulnerability in the code that handles the token. This could include a "Buffer Overflow", "XSS", "Denial of Service",

"Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", or "SQL injection" attack among many others. The content of the manipulated token should be reviewed to better understand what type of attack the user was attempting, however because the tokens are heavily encrypted and validated, this incident does not represent a threat to the security of the system tracking mechanism.

Tracking Processors: Client Beacon Processor: Incident - Beacon Session Tampering

Complexity: Medium (3.0)

Default Response: 1x = 5 day Clear Inputs in 10 minutes.

Cause: WebApp Secure uses a special persistent token that inserts itself in multiple locations throughout the client. When a user returns to the site later on, these tokens are transmitted back to the server. This allows the server to correlate the traffic issued by the same user, even if the requests are weeks apart. This incident is triggered when the user manipulates the token data being transmitted to the server on a subsequent visit. They manipulated the data in such a way as to remain consistent with the correct formatting for the token, but the token itself is not valid and was never issued by the server.

Behavior: Attempts to manipulate and spoof the tracking tokens are generally performed when the attacker is trying to figure out what the token is used for and potentially evade tracking. If they are assuming it's used for session management, this might also be a part of a "Credential/Session Prediction" attack. Because the format of the submitted modified token is still consistent with the format expected, this is not likely a generic input attack. It also does not represent any threat to the system, as the modified token is simply ignored.

Tracking Processors: Client Fingerprint Processor

This processor is designed to collect uniquely identifying information from requests issued by a user. This information is then compared to the information collected about other sessions in the system. If a match is identified, the two sessions are merged. This allows session association to work even if all storage mechanisms used by the other tracking processors are cleared. Some of the uniquely identifying information includes the browser plugin list, the system font list, time skew, time-zone, user-agent, system language, etc.

Table 27: Client Fingerprint Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	False	Whether traffic should be passed through this processor.
Exclude Rules	Collection	[collection:0]	The fingerprint association rules to ignore.
Excluded Collectors	Collection	[collection:0]	The data points to prevent collection of on the client.

Table 27: Client Fingerprint Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Hash Fingerprint Data	Boolean	False	Whether to hash the raw fingerprint data points before storing them. This prevents the recorded data from being used to obtain the original information about the client and reduces the overall storage size requirements. If collecting PII data is a concern, this is a recommended option, as it will eliminate any PII data in place of hashed versions of that data which cannot be reversed.
Page Injection Enabled	Boolean	True	Whether the fingerprint script should be injected into the requested page.
Advanced			
Binary Resource Directory	String	(randomized)	The fake directory where binary resources required by the fingerprinting script are served from.
Data Obfuscation Key	String	(randomized)	The key used to prevent easy reading of the submitted fingerprint data. This should be alphanumeric and at least 8 unique characters long, duplicate characters are allowed, but do not count toward the total 8.
Fingerprint Scope Key	String	(randomized)	The key used to store fingerprint data. If this key is changed, all previously stored fingerprint data will be lost and the system will begin collecting fresh fingerprint data.
Fingerprint Submission Response	HTTP Response	text/plain 200 OK	The response to return when a user attempts to submit a fingerprint in the background. The user will not see this response unless they are using a debug proxy.

Table 27: Client Fingerprint Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Fingerprint Tracking Cookie Name	String	(randomized)	The name of the cookie used on the client to ensure we don't submit multiple copies of the same fingerprinting data. This can be anything, but should not overlap with a legitimate cookie being used on the site.
Hash Fingerprint Data	Boolean	False	Whether to hash the raw fingerprint data points before storing them. This prevents the recorded data from being used to obtain the original information about the client and reduces the overall storage size requirements. If collecting PII data is a concern, this is a recommended option, as it will eliminate any PII data in place of hashed versions of that data which cannot be reversed.
Script Filename	String	(randomized)	The filename to use when serving the fingerprint script to the client.
Submission Filename	String	(randomized)	The filename where fingerprint data should be submitted back to the server
Incident: Fingerprint Directory Indexing	Boolean	True	The user requested a directory index listing on the fake directory used to serve binary resources required by the fingerprinting script. Since this is a fake directory, the request represents a malicious action.
Incident: Fingerprint Directory Probing	Boolean	True	The user requested a random file within the fake directory used to serve binary resources required by the fingerprinting script. Since only files we specifically reference in the fingerprinting script should be requested, this represents a malicious action.

Table 27: Client Fingerprint Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Incident: Fingerprint Manipulation	Boolean	True	The user submitted fingerprint data to the server which was not properly formatted. This likely means that the user was manipulating the fingerprinting data or spoofed it entirely.

Tracking Processors: Client Fingerprint Processor: Incident - Fingerprint Directory Indexing

Complexity: Low (2.0)

Default Response: n/a

Cause: The client fingerprint processor is designed to obtain a semi-unique identifier from the clients rendering engine. The fingerprint is a hash of data obtained through JavaScript such as the plugin list, time zone, and screen resolution. In order to calculate a fingerprint, some binary resources such as flash objects may be required. These resources will be served from a known fake directory. This incident is triggered if the user attempts to get a directory index listing from the known fake resource directory.

Behavior: If an attacker discovers the script being used to collect and submit the fingerprint data, they may be interested to know what else is in the directory where fingerprint binary resources are served. As such, they may request a directory index listing from the fake directory. Because the directory is fake, there are no files to list, but the simply action of attempting to get the list is indicative of abusive behavior. If an attacker is able to obtain a directory index listing, they may attempt to exploit some of the other resources in the directory, or gain information about the web site that may otherwise not be available. Any attempts to index the directory will result in a 403, which will yield no useful information to the attacker. This is usually part of a spidering effort and targets "Predictable Resource Location" vulnerabilities.

Tracking Processors: Client Fingerprint Processor: Incident - Fingerprint Directory Probing

Complexity: Low (2.0)

Default Response: n/a

Cause: The client fingerprint processor is designed to obtain a semi-unique identifier from the clients rendering engine. The fingerprint is a hash of data obtained through JavaScript such as the plugin list, time zone, and screen resolution. In order to calculate a fingerprint, some binary resources such as flash objects may be required. These resources will be served from a known fake directory. This incident is triggered if the user attempts to request a file in the fake directory that does not exist. In other words, they are looking for a specific file that does not exist within a fake directory.

Behavior: If an attacker discovers the script being used to collect and submit the fingerprint data, they may be interested to know what else is in the directory where fingerprint resources are served from. As such, they may request specific files they think they be inside the fake directory. Because the directory is fake, there are no actual files available, but the simply action of attempting to get a resource that does not exist in a fake directory is indicative of abusive behavior. This type of attack is generally targeted at "Predictable Resource Location" vulnerabilities.

Tracking Processors: Client Fingerprint Processor: Incident - Fingerprint Manipulation

Complexity: Medium (3.0)

Default Response: n/a

Cause: The client fingerprint processor is designed to obtain a semi-unique identifier from the clients rendering engine. The fingerprint is a hash of data obtained through JavaScript such as the plugin list, time zone, and screen resolution. This incident is triggered when the user attempts to submit an invalid fingerprint.

Behavior: Normally, the fingerprinting code will be allowed to execute on the client without any problems. However if an attacker discovers the fingerprinting code, they may try to spoof fingerprints of other users, or simply try to exploit the fingerprint service. To do this, they may create a fake fingerprint value and submit it to the server in the same way that legitimate fingerprints are submitted. It likely would not be clear to the attacker as to what the value is used for, or how the value is consumed by the server, so this type of activity would be purely exploratory. Once the attacker identifies a valid fingerprint that was not generated from their rendering engine, they will likely continue to statically submit that same fingerprint on all transactions. Once that happens, it will not be possible to identify the manipulated fingerprint. So this incident should come early in the attack, but will stop once the attacker has reached their goal. In such a case, the attacker is simply trying to disguise their true identity. If the modified fingerprint is not alpha numeric and contains special characters, then the attacker is probably attempting to launch a targeted attack against the way the service consumes the data, such as a "SQL Injection", "XSS", or "Buffer Overflow" attack.

Tracking Processors: Client Classification Processor

The client classification processor is designed to detect popular legitimate search engine bots. These types of bots are notorious for performing aggressive spider activity on web sites, and often this activity can trigger security related incidents. Using this processor to define the conditions used to identify such bots, allows the system to ignore security incidents from those clients. This will remove search engine related false positives, as well as prevent errors in indexed and cached results. The popular search engines are included by default, but if additional search engines should be allowed, new rules can be created. Be careful not to define a rule that will match clients other than the targeted search engine bot. The less specific the conditions of a rule, the easier it will be for an attacker to spoof the search engine and circumvent detection. It is critical that DNS be enabled on WebApp Secure to achieve effective classification of search engines. Not enabling DNS and leaving this processor turned on, may result in some attackers not being identified.

If a client is classified as a search engine based on one of the defined rules, then that client will not be able to generate incidents, and additionally:

- Query String Processor will be turned off for that user (no query param injections)
- Hidden Link Processor will be turned off for that user (no hidden link injections)

This is done to ensure that the results cached by the search engine bot do not include fake code that may change in the future, and thus end up flagging clients who are following legitimate search engine links. Classification rules are made up of a series of patterns to run against various attributes of the client:

- IP Address
- Hostname
- User Agent
- Country Code
- City
- Region
- Header Name and Value

At least one pattern must be specified on at least one attribute, however you can specify patterns for as many attributes as the bot will allow. For example, if the bot changes its IP address constantly, then you should not define a pattern for the IP. However if the hostname always ends in google.com, then a pattern of `[.]google[.]com$` could be assigned to the "Hostname" attribute. If the user agent always contains "googlebot", then "googlebot" could be assigned as the user agent pattern. Here is an example of a complete pattern for the Googlebot search engine spider:

Hostname Pattern: `[.]google(bot)?[.]com$`
 User Agent Pattern: `(adsbot.google|googlebot|Google[]Web[]Preview|Mediapartners-Google)`
 Country Pattern: `US`
 Region Pattern: `(California|Georgia)`



NOTE: It would be extremely difficult for an attacker to spoof values for all of those attributes which would match the patterns. For example, spoofing the reverse DNS lookup to end in ".google.com" would require serious effort, and would require insecure DNS configuration on behalf of the WebApp Secure administrator. Ideally every rule should include either an "ip" or "hostname" pattern.

Table 28: Client Classification Configuration Parameters

Parameter	Type	Default Value	Description
-----------	------	---------------	-------------

Basic

Table 28: Client Classification Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Processor Enabled	Boolean	False	Whether traffic should be passed through this processor.
Classification Rules			
Client Type	String	(none)	The name of the type of client being identified.
IP Pattern	String	(none)	The IP address pattern to require (if any).
Hostname Pattern	String	(none)	The hostname pattern to require (if any) if DNS is enabled.
User Agent Pattern	String	(none)	The user agent pattern to require (if any).
Country Pattern	String	(none)	The country pattern to require (if any).
City Pattern	String	(none)	The city pattern to require (if any).
Region Pattern	String	(none)	The region pattern to require (if any).
Header Name Pattern	String	(none)	A pattern used to identify a required header name (if any).
Header Value Pattern	String	(none)	A pattern used to verify the value of a header that matches the header name pattern (if any).

CHAPTER 32

Response Processors

- [Response Processors on page 221](#)
- [Response Processors: Block Processor on page 222](#)
- [Response Processors: Request Captcha Processor on page 223](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Answer Automation on page 226](#)
- [Response Processors: Request Captcha Processor: Incident - No Captcha Answer Provided on page 227](#)
- [Response Processors: Request Captcha Processor: Incident - Multiple Captcha Request Overflow on page 228](#)
- [Response Processors: Request Captcha Processor: Incident - Unsupported Audio Captcha Requested on page 228](#)
- [Response Processors: Request Captcha Processor: Incident - Bad Captcha Answer on page 229](#)
- [Response Processors: Request Captcha Processor: Incident - Mismatched Captcha Session on page 230](#)
- [Response Processors: Request Captcha Processor: Incident - Expired Captcha Request on page 230](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Request Tampering on page 231](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Signature Tampering on page 232](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Signature Spoofing on page 233](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Cookie Manipulation on page 233](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Image Probing on page 234](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Request Size Limit Exceeded on page 235](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Disallowed MultiPart on page 236](#)

- [Response Processors: Request Captcha Processor: Incident - Captcha Directory Indexing on page 236](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Directory Probing on page 237](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Parameter Manipulation on page 238](#)
- [Response Processors: Request Captcha Processor: Incident - Captcha Request Replay Attack on page 239](#)
- [Response Processors: Request Captcha Processor: Incident - Multiple Captcha Replays on page 240](#)
- [Response Processors: Request Captcha Processor: Incident - Multiple Captcha Disallow Multipart on page 241](#)
- [Response Processors: Request Captcha Processor: Incident - Multiple Captcha Parameter Manipulation on page 242](#)
- [Response Processors: CSRF Processor on page 243](#)
- [Response Processors: CSRF Processor: Incident - CSRF Parameter Tampering on page 245](#)
- [Response Processors: CSRF Processor: Incident - Multiple CSRF Parameter Tampering on page 246](#)
- [Response Processors: CSRF Processor: Incident - CSRF Remote Script Inclusion on page 247](#)
- [Response Processors: CSRF Processor: Incident - HTTP Referers Disabled on page 247](#)
- [Response Processors: Header Injection Processor on page 248](#)
- [Response Processors: Force Logout Processor on page 248](#)
- [Response Processors: Strip Inputs Processor on page 249](#)
- [Response Processors: Slow Connection Processor on page 249](#)
- [Response Processors: Warning Processor on page 250](#)
- [Response Processors: Warning Processor: Incident - Warning Code Tampering on page 251](#)
- [Response Processors: Application Vulnerability Processor on page 252](#)
- [Response Processors: Application Vulnerability Processor: Incident - App Vulnerability Detected on page 252](#)
- [Response Processors: Support Processor on page 253](#)
- [Response Processors: Cloppy Processor on page 254](#)
- [Response Processors: Login Processor on page 255](#)
- [Response Processors: Login Processor: Incident - Site Invalid Login on page 261](#)
- [Response Processors: Login Processor: Incident - Site Login Multiple IP on page 262](#)
- [Response Processors: Login Processor: Incident - Site Login Multiple Usernames on page 262](#)
- [Response Processors: Login Processor: Incident - Site Login User Sharing on page 263](#)

- [Response Processors: Login Processor: Incident - Site Login User Pooling on page 263](#)
- [Response Processors: Login Processor: Incident - Site Login User Brute Force on page 264](#)
- [Response Processors: Login Processor: Incident - Site Login Brute Force on page 264](#)
- [Response Processors: Login Processor: Incident - Site Login Username Scan on page 264](#)
- [Response Processors: Google Map Processor on page 265](#)

Response Processors

The processors in this section are responsible for issuing the various counter responses to malicious users on a server protected by WebApp Secure. A response is activated when WebApp Secure believes intervention is required between the profiled user and the webserver. This response can manifest into any of the types fully explained below.

Response Methodology: When WebApp Secure believes a response is required, the type of response issued depends on the type of behavior the malicious user exhibited to receive the response. For example, users that WebApp Secure think are automated tools will likely get issued a CAPTCHA response, whereas it is obvious that a real malicious user (not a bot) will be able to solve a CAPTCHA. In the second case, adding a 2 to 6 second slow might be more effective at wasting the hacker's time. Another factor that comes into play when issuing counter responses is risk level. If WebApp Secure believes a user is of no immediate risk to the system, it might only activate those responses which still allow the user to browse the site somehow, such as the Warning response or Slow Connection response. This way, WebApp Secure can monitor that user and gather additional information to properly assess their risk level. If WebApp Secure believes the user is a danger to the system, it will issue a more severe response, such as stripping out all inputs on every request or outright blocking the profile. Some responses might not get issued right away. For example, an incident may produce "a permanent block in 20 minutes". The reason for this delay in the counter response is that WebApp Secure uses this buffer time to gather some last-minute information on the profile before issuing the final response. WebApp Secure will respond instantly if it perceives immediate threat to the integrity of the system, but instances where this is not the case allow WebApp Secure to profile the attacker for a bit longer. The end result will be a more complete look at the attacker and his/her habits.

Types of Responses: Certain response processors are self-explanatory, such as the Block Processor (the user will see that they are blocked). Other responses are "invisible" in that there are no manifestations of the response visible to the user. An example of an invisible response processor is the Strip Inputs Processor. This processor will simply Block Processor 125 remove all values from all inputs on any form submitted because WebApp Secure has determined that the user's input can no longer be trusted. On the user-end, they will see nothing that will indicate to them that this response is active (until they figure out that all inputs are not being recognized).

Response Activation: Responses get automatically activated according to rules set forth within WebApp Secure. These rules are outlined for each incident a user can trigger, and are described in the documentation for each processor. The default response for each incident is documented in the User Guide, and will look something like, "Default Response: 1x = Warn User. 2x = 1 Day Block". The '1x' or '2x' indicate the number of incidents of that

type triggered. For this example, triggering this incident once results in the Warning Processor being activated. If the same incident is triggered again on the same profile, the user then gets a 1 day block via the Block Processor.



NOTE: You may wish to completely disable automatic counter responses entirely. If this is the case, changing the configuration parameter **Auto Response Activation Enabled** to **False** will prevent any new automatic activations, but will not hinder your ability to manually activate responses on profiles. (Configuration > Global Configuration > Auto Response Service > Auto Response Activation Enabled = False)

Compounding and Overriding Responses

- Warning - There is no need to warn someone when they are already blocked.
- Captcha - If the user is ever unblocked (or the block expires), they will be prompted to solve the captcha.
- Cloppy - If they are ever unblocked (or the block expires) Cloppy will appear.
- Google Maps - If the user is ever unblocked (or the block expires), they will be shown the Google map.

Captcha overrides:

- Warning - WebApp Secure will warn after they solve the captcha.
- Cloppy - Cloppy will appear after they solve the captcha.
- Google Maps - The Google map will be shown after they solve the captcha.

Strip Inputs overrides:

- Break Authentication - It is redundant, as WebApp Secure is already stripping login credentials.

Response Processors: Block Processor

The block processor is actually a form of auto response. When this processor is enabled, it will allow the security system to block a response with "Blocked!" message sent back to the user.



NOTE: There are no actual triggers for this processor; it is a form of response.

Table 29: Block Processor Configuration Parameters

Parameter	Type	Default Value	Description
-----------	------	---------------	-------------

Basic

Table 29: Block Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Block Response	Configurable	HTTP Response	The response to return to the user when they are blocked.

Response Processors: Request Captcha Processor

The Captcha processor is designed to protect specific pages in a web application from automation. This is done by using a "Captcha" challenge, where the user is required to transcribe random characters from an obscured image or muffled audio file in order to complete the request. The intent is that a human would be capable of correctly answering the challenge, while an automated script with no human intervention would be unable to do so. This assumes that the image is obscured enough that text recognition software is not effective, and the audio file significantly distorted to defeat speech-to-text software. Requiring such user interaction is somewhat disruptive, so it should be utilized only for pages that are prime automation targets (such as contact forms, registration pages, login pages, etc.). Furthermore, these captcha challenges can be customized to fit the style of the application it is protecting.

Table 30: Request Captcha Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Protected Pages	Collection	None	A collection of protected pages.
Advanced			
Bad Request Block Response	HTTP Response	400 HTTP Response	The response to return if the user issues a request that either is too large, or uses multipart and multi-part is disabled.
Blocked Replay Response	String	Random Value	The response to return if the user attempts to submit the validated request multiple times using the same captcha answer, and that behavior is not allowed.
Captcha Binary Directory	String	Random Value	The name of the directory where captcha images and audio files will be served from. This should not conflict with any actual directories on the site.

Table 30: Request Captcha Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Captcha Characters	String	Random Value	The characters to use when generating a random captcha value. Avoid using characters that can be easily mixed up. This set of characters is case sensitive.
Captcha State Cookie Name	String	Random Value	The name of the cookie to use to track the active captchas that have not yet been solved. The cookie is only served to the captcha binary directory.
Captcha Validation Input Name	String	Random Value	The name of the form input used to transmit the captcha validation key. This should be obscure so that users who have not been required to enter a captcha cannot supply bad values to this input to profile the system.
Maximum Active Captchas	Integer	7	The maximum number of captchas any given user can be solving at any given time. This limit can be overcome, but the majority of users will not be able to. This is primarily for performance, as the more active captchas that are allowed, the larger the state cookie becomes.
Support Audio Version	Boolean	True	Whether an audio version of the captcha is provided to the user. This may be a requirement for accessibility, as vision impaired users would otherwise be unable to solve the captcha.
Watermark	String	Random Value	The text to watermark the captcha with. This can be used to prevent the captcha from being used in a phishing attack. For example, an abuser would not be able to simply display the captcha on a different site and ask a user to solve it. The watermark would tip the user off that the captcha was not intended for the site they are visiting. Use %DOMAIN to use the domain name as the watermark.
Cancel URL	String	None	The URL to redirect the user to if they cancel the captcha. This should not be to the same domain, because the domain is being blocked using a captcha, and therefore, canceling would only redirect to a new captcha. An empty value will hide the cancel button.
Captcha Expiration	Integer	2 minutes	The maximum number of seconds the user has to solve the captcha before the request is no longer possible.
Expired Captcha Response	HTTP Response	400 HTTP Response	The response to return if the user submits a validated request after the captcha has expired. This may happen if the user refreshes the results of the captcha long after they have solved it.
Maximum Request Size	Integer	500kb	The maximum number of bytes in a request before it is considered not acceptable for captcha validation, and will be blocked.
Incident: Bad Captcha Answer	Boolean	False	The user was asked to solve a captcha and entered the wrong value. This could be a normal user error, or it could be the results of failed abuse.

Table 30: Request Captcha Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Incident: Captcha Cookie Manipulation	Boolean	True	The user submitted a request and was asked to solve a captcha. They then modified the state cookie used to track captchas, making it invalid. This is likely in an attempt to find a way to bypass the captcha validation mechanism.
Incident: Captcha Directory Indexing	Boolean	True	The user has requested a directory index in the directory that serves the captcha images and audio files. This is likely in an attempt to get a list of all active captchas or to identify how the captchas are generated.
Incident: Captcha Directory Probing	Boolean	True	The user has requested a random file inside the directory that serves the captcha images and audio files. This is likely in an attempt to find an exploitable service or sensitive file that may help bypass the captcha validation mechanism.
Incident: Captcha Disallowed MultiPart	Boolean	True	The user has submitted a multipart form post to the protected page, which has been configured as a disallowed option. This is likely in an attempt to find an edge case the captcha validation mechanism is not expecting.
Incident: Captcha Image Probing	Boolean	True	The user is probing the directory used to serve captcha images. This is likely in an attempt to find hidden files or a way to invoke errors from the captcha serving logic.
Incident: Captcha Parameter Manipulation	Boolean	True	The user has submitted a request with a valid captcha, but they modified the query string parameters. This could be in an attempt to change the output of executing the request without requiring the user to re-validate with another captcha.
Incident: Captcha Request Replay Attack	Boolean	True	The user has attempted to submit the same request multiple times with the same captcha answer. In other words, they solved the captcha once and issued the resulting request multiple times.
Incident: Captcha Request Size Limit Exceeded	Boolean	True	The user has submitted a request to the protected page which contains more data than is allowed. This is may be an attempt to reduce system performance by issuing expensive requests, or it may be an indicator of a more complex attack.
Incident: Captcha Request Tampering	Boolean	True	The user submitted a request and was asked to solve a captcha. They introspected the page containing the captcha and altered the serialized request data (the data from the original request before the captcha prompt). They then submitted a valid captcha using the modified request data. This is likely in an attempt to abuse the captcha system and identify a bypass technique.
Incident: Captcha Signature Spoofing	Boolean	True	The user submitted a request and was asked to solve a captcha. They introspected the page containing the captcha and provided a validation key from a previously solved captcha. This is likely in an attempt to submit multiple requests under the validation of the first.

Table 30: Request Captcha Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Incident: Captcha Signature Tampering	Boolean	True	The user submitted a request and was asked to solve a captcha. They introspected the page containing the captcha and provided a fake validation key. This is likely in an attempt to bypass the captcha validation mechanism.
Incident: Expired Captcha Request	Boolean	True	The user submitted a request and was given a set window of time to solve a captcha. The user solved the captcha and submitted the request for final processing after the window of time expired. This is likely an indication of a packet replay attack, where the user attempts to invoke the business logic of the protected page multiple times under the same captcha validation.
Incident: Mismatched Captcha Session	Boolean	True	The user submitted a request and was asked to solve a captcha. They solved the captcha, but upon submitting the request for final processing, they did so under a different session ID. This is likely due to multiple machines participating in the execution of the site workflow and may indicate a serious targeted automation attack.
Incident: No Captcha Answer Provided	Boolean	True	The user attempted to validate a captcha but did not supply an answer to validate. There is no interface that allows the user to do this, so they must be manually executing requests against the captcha validation API in an attempt to evade the mechanism.
Incident: Unsupported Audio Captcha Requested	Boolean	True	The user has requested an audio version of the captcha challenge, but audio is not supported and there should not be an interface to ask for the audio version. The user is likely trying to find a way to more easily bypass the captcha system.

Response Processors: Request Captcha Processor: Incident - Captcha Answer Automation

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds and 1 Day Clear Inputs

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve

"Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides an abnormal volume of bad solutions to the captcha image. For example, the image may have said "Hello", but the user attempted 30 different values all of which did not match "Hello". Because the images can be somewhat difficult to read at times (in order to ensure a script cannot break them), it is not uncommon for a legitimate user to enter the wrong value a few times before getting it right, especially if they are unfamiliar with this type of technique, but after dozens of failed attempts, it is more likely a malicious user.

Behavior: Simply providing a bad solution to the captcha image is not necessarily malicious. Legitimate users are not always able to solve the captcha on the first try. However if a large volume of invalid solutions are provided, then it is more likely that a script is attempting to crack the captcha image through educated guessing and "Brute Force".

Response Processors: Request Captcha Processor: Incident - No Captcha Answer Provided

Complexity: Medium (3.0)

Default Response: 1x = Warn User. 2x = 1 Day Block

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user forces the captcha interface to submit the request without a valid captcha solution. There is no way to do this without manipulating the logic that controls captcha protected requests.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. In this specific case, the attacker attempted to submit the captcha protected page without actually solving the captcha. Instead they provided an empty value for the solution parameter. It is not possible to submit an empty solution using the provided captcha interface, so this is almost guaranteed to be a malicious attempt at

generating an error and obtaining additional details about the captcha implementation though an "Information Leakage" weakness.

See <http://projects.webappsec.org> for information on attack types.

Response Processors: Request Captcha Processor: Incident - Multiple Captcha Request Overflow

Complexity: Low (2.0)

Default Response: 1x = 1 Day Clear Inputs.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit dozens of captcha protected requests that exceed the configured maximum for protected request sizes.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. In this specific case, the attacker submitted dozens of extremely large requests, probably in an effort to find a "Buffer Overflow" vulnerability, which would produce useful error data and potentially open the server up to further exploitation. They may also be attempting to overload the server and execute a "Denial of Service" attack.

Response Processors: Request Captcha Processor: Incident - Unsupported Audio Captcha Requested

Complexity: Medium (3.0)

Default Response: 3x = Slow Connection 2-6 seconds and Warn User. 5x = 1 Day Block.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed

to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to request the audio version of a captcha challenge when support for audio captchas has been explicitly disabled.

Behavior: Solving an image based captcha is exceptionally difficult and requires a great deal of time and research. Solving an audio captcha however is far less difficult. There are already multiple open source libraries available for translating speech to text. As such, it is often necessary to disable the support of "audio" captchas for critical workflows (such as administrative login dialogs), unless absolutely necessary for accessibility reasons. This incident occurs when the audio captcha has been disabled, but a user is attempting to manually request the audio version of the captcha challenge anyway. The captcha interface does not expose a link to the audio version unless it is explicitly enabled in configuration, so this would require that the user knows where to look for the audio version, they understand the filename conventions, and they know how to make the request manually to download the file. In either case, if audio captchas are not enabled (through configuration), then this effort will not be successful.

Response Processors: Request Captcha Processor: Incident - Bad Captcha Answer

Complexity: Suspicious (1.0)

Default Response: 10x = Captcha Answer Automation Incident.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a bad solution to the captcha image. For example, the image may have said "Hello", but the user typed "hfii0" instead. Because the images can be somewhat difficult to read at

times (in order to ensure a script cannot break them), it is not uncommon for a legitimate user to enter the wrong value a few times before getting it right, especially if they are unfamiliar with this type of technique.

Behavior: Simply providing a bad solution to the captcha image is not necessarily malicious. Legitimate users are not always able to solve the captcha on the first try. However if a large volume of invalid solutions are provided, then it is more likely that a script is attempting to crack the captcha image through educated guessing and "Brute Force".

Response Processors: Request Captcha Processor: Incident - Mismatched Captcha Session

Complexity: High (4.0)

Default Response: 1x = Warn User, 2x = 5 Day Clear Inputs.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a solution to a captcha that was issued for a different session than their own, as might be the case in a script that uses minimal human interaction to solve the captcha's, but everything else is automated

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try and harvest successfully solves captchas from other users on the site. This can be done either by infecting those machines with a virus, or by implanting script into some of the sites pages (possibly through XSS). If this technique is used, then the captcha that is being solved may not have originated from the same session as the user who is submitting the solution. This is a dead giveaway that the user is attempting to defeat the captcha system to automate a specific task.

Response Processors: Request Captcha Processor: Incident - Expired Captcha Request

Complexity: Suspicious (1.0)

Default Response: None.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a solution to a captcha after the allotted time for solving the captcha has elapsed.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to run expensive image processing algorithms on the captcha image in order to identify what the represented value might be. Additionally, a user might attempt to send the captcha to a warehouse of human captcha solvers. These warehouses specialize in solving large volumes of captchas at a fairly low price (less than a penny per captcha). In either case, it can take several minutes to get the correct captcha answer, and will likely run out the amount of time the user is allowed for solving the captcha. If using a browser, the input would flat out stop accepting answers, but in a scripted scenario, the script will likely try and submit the value anyway, because it is unaware of the expiration. It is possible that this incident would be triggered by a legitimate user, if they were to refresh the page that was produced after the captcha was solved. This would effectively cause the captcha to be reprocessed after the expiration time had been exceeded. As such, this incident on its own is not considered malicious.

Response Processors: Request Captcha Processor: Incident - Captcha Request Tampering

Complexity: High (4.0)

Default Response: 1x = Warn User. 2x = 5 Day Clear Inputs.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site

(similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a solution to a captcha which is correct, but they have modified the parameter containing the original request (which is heavily encrypted to prevent tampering).

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. The parameter that was modified contained the original request data (before the captcha was issued), it is likely that the attacker is attempting to smuggle a malicious payload through the system without being detected by any network or web firewalls. Because this parameter uses heavy encryption and validation, this type of activity will not produce any useful information or expose any vulnerabilities. Depending on the value they submitted for the original request data, this may also fall under one of the other attack categories involving manipulating general inputs, such as a "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", or "SQL injection" attack among many others.

Response Processors: Request Captcha Processor: Incident - Captcha Signature Tampering

Complexity: High (4.0)

Default Response: 1x = Warn User. 2x = 5 Day Clear Inputs.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a solution to a captcha which is correct, but they have modified the integrity checking signature passed along with the captcha solution.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha

mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. Depending on the value they submitted for the original request data, this may also fall under one of the other attack categories involving manipulating general inputs, such as a "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", or "SQL injection" attack among many others.

Response Processors: Request Captcha Processor: Incident - Captcha Signature Spoofing

Complexity: High (4.0)

Default Response: 1x = Warn User. 2x = 5 Day Clear Inputs.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a solution to a captcha which is correct, but they have replaced the integrity checking signature passed along with the captcha solution to one that was used in a previous captcha solution.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. This specific incident generally reflects the behavior of a user who is trying to submit a request that would normally be protected by a captcha, but they are trying to trick the system into thinking the captcha was solved correctly, even though it was not. This is generally looking for a "Insufficient Anti-Automation" weakness in the captcha handling mechanism.

Response Processors: Request Captcha Processor: Incident - Captcha Cookie Manipulation

Complexity: Medium (3.0)

Default Response: 1x = Warn User. 2x = 5 Day Clear Inputs.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user alters the cookies used to maintain captcha state.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. In this specific case, the attacker modified a cookie that is used to maintain the state of the captcha. The cookie is heavily encrypted, but the attacker may be attempting to establish a way of either identifying what the value of the captcha is algorithmically (by analyzing the cookie value), or they may be attempting to assign a value to the captcha. In either case, this activity generally indicates a user who is trying to find a way to bypass the captcha. Depending on the value they submitted for the original request data, this may also fall under one of the other attack categories involving manipulating general inputs, such as a "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", or "SQL injection" attack among many others.

Response Processors: Request Captcha Processor: Incident - Captcha Image Probing

Complexity: Low (2.0)

Default Response: 1x = Warn User. 2x = 5 Day Block.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web

application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to request a captcha image file for a request that is not being protected by a captcha.

Behavior: In order to find a way to bypass the captcha mechanism, attackers will often attempt to collect a large number of captcha images for offline analysis. If the attacker can find a pattern in how the captcha images are issued, or how the filename relates to the value in the image, then they can effectively bypass the captcha mechanism at will. In this case, the attacker is guessing arbitrary captcha image filenames, but is attempting to keep the format of the names consistent with known captcha image URL's. Because the filename used and the values in the image have no correlation, this technique will not be successful and will simply waste the attacker's time and resources.

Response Processors: Request Captcha Processor: Incident - Captcha Request Size Limit Exceeded

Complexity: Suspicious (1.0)

Default Response: 10x = Multiple Captcha Request Overflow Incident.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit a captcha protected request that contains a request body larger than the configured maximum.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated or if an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. In this specific case, the attacker submitted an extremely large request, probably in an effort to find a "Buffer Overflow" vulnerability, which would produce useful error data and potentially open the server up to further exploitation. This incident is not

necessarily malicious on its own, as it is possible for a normal user to submit a value that is larger than the configured maximum, especially if the configured maximum is small, or if the form protected by the captcha allows file posts.

Response Processors: Request Captcha Processor: Incident - Captcha Disallowed MultiPart

Complexity: Suspicious (1.0)

Default Response: 10x = Multiple Captcha Disallow Multipart Incident.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit a captcha protected request that contains a binary file, and the captcha is explicitly configured to not allow binary file submission (it has been configured to disallow multi-part form submissions).

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. In this specific case, the attacker submitted a binary file in the request that is being protected. The captcha in this case has been explicitly configured to not allow Multi-Part form submissions, so this represents unexpected and undesired activity. Using Multi-Part forms, the attacker can more easily accomplish a "Buffer Overflow" attack, which would produce useful error data and potentially open the server up to further exploitation. Additionally, some web applications do not handle the encoding used for multi-part forms gracefully, so error information may also be obtained from conflicts arising from the submission type. This is not necessarily a malicious incident on its own, because it is possible that the user is legitimately submitting a multi-part form, and just happened to have the captcha activated during the submission. However this is a very rare case, and still represents a somewhat suspicious client.

Response Processors: Request Captcha Processor: Incident - Captcha Directory Indexing

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds and 1 Day Block.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to request a directory index from the same fake directory as the captcha images are being served from.

Behavior: When attempting to either bypass the captcha mechanism, or find a vulnerability in the server, attackers will often try finding unlinked resources throughout the web site. The captcha mechanism uses a fake directory in order to serve the images and audio files that contain the captcha challenge. If the attacker is requesting an arbitrary file within the same fake directory, they are likely trying to find a "Predictable Resource Location" vulnerability. In this specific case, the attacker is attempting to get a full file listing of everything inside the captcha directory. This could potentially be used to get a massive list of all active captcha URL's, or to find resources that are used in the creation of captcha challenges. The directory index will not be allowed, so this does not actually provide the attacker with any useful information.

Response Processors: Request Captcha Processor: Incident - Captcha Directory Probing

Complexity: Low (2.0)

Default Response: 1x = Warn User. 2x = Slow Connection 2-6 seconds and 5 Day Block.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve

"Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to request an arbitrary file (not a captcha image, but something else) from within the same fake directory as the captcha images are being served from.

Behavior: When attempting to either bypass the captcha mechanism, or find a vulnerability in the server, attackers will often try finding unlinked resources throughout the web site. The captcha mechanism uses a fake directory in order to serve the images and audio files that contain the captcha challenge. If the attacker is requesting an arbitrary file within the same fake directory, they are likely trying to find a "Predictable Resource Location" vulnerability. For example, the attacker might be trying to find a source file in the captcha serving directory in hopes of actually being able to get the source code behind how captcha images are generated. Because the directory is fake, the attacker will never find any of the resources they are looking for.

Response Processors: Request Captcha Processor: Incident - Captcha Parameter Manipulation

Complexity: Suspicious (1.0)

Default Response: 5x = Multiple Captcha Parameter Manipulation Incident.

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit multiple solutions for multiple captchas, but they keep modifying the query parameters that were submitted with the original requests. For example, if the user submitted a "add product to cart" request, and one of the query parameters was the item to add, this incident would be triggered if after solving the captcha, the value of that query parameter was modified to some other value, and this modification happened dozens of times.

Behavior: Because captcha's prevent automation, attackers will sometimes try and find ways to abuse the technique used to request the captcha in order to exploit the site. For example, if the attacker can find a way to submit the same solution over and over again, but have the web application perform a different action each time, they may be able to solve the captcha once and still automate the resulting workflow. In this case, the attacker changed a query parameter that was submitted with the original request. They submitted

the original request, solved the captcha, changed the query parameter, and then resubmitted the solved captcha request. In some cases, this might cause the web application to execute a different operation based on the difference in query parameter values. For example, if the protected workflow is "add product to cart" on a shopping site, then the attacker might attempt to submit the same solved captcha repeatedly, but change the product ID that is being added on each request. This might allow them to automate the addition of products to a shopping cart, after solving only one captcha challenge. The captcha mechanism does not allow the modification of query parameters after the original request has been submitted, so this type of activity will not be successful.

Response Processors: Request Captcha Processor: Incident - Captcha Request Replay Attack

Complexity: Suspicious (1.0)

Default Response: 5x = Multiple Captcha Replay Incident

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit a captcha solution multiple times and "replay" is explicitly disabled for the captcha being used.

Behavior: Because captcha's prevent automation, attackers will sometimes try and find ways to abuse the technique used to request the captcha in order to exploit the site. For example, if the attacker can find a way to submit the same solution over and over again, they may be able to solve the captcha once and still automate the resulting workflow. This is sometimes considered legitimate behavior (as would be expected if the user refreshed the browser after submitting a successful captcha), however in many cases, such functionality would make the captcha significantly less effective at preventing automation. In this case, the attacker resubmitted a request that had already been successfully validated through a captcha, and "replay" was explicitly disabled for the captcha. This is not necessarily a malicious incident on its own, because the user may have accidentally refreshed the browser, however multiple attempts would definitely represent malicious intent. An example of where a captcha's "replay" could cause a problem is on a gaming site, where the user is adding fake "money" to their account. In order to add the fake money, they must solve the captcha. This workflow is protected with a captcha, because if a user could automate the process, they would be able to add

unlimited funds to their account. If an attacker were able to solve the captcha once, and continuously resubmit the resulting request, they could effectively add funds over and over again without resolving a new captcha. This would then allow for automation. Replay attackers are less of a problem if the web application being protected already has a method of preventing the same request from being submitted accidentally multiple times. Such would be the case if the web application maintained state information for the given session, and recorded the operation after it was successful, then used that state information to prevent a future occurrence of the operation.

Response Processors: Request Captcha Processor: Incident - Multiple Captcha Replays

Complexity: Low (2.0)

Default Response: 1x = Warn User, 2x = 1 Day Clear Inputs

Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by the system. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit a captcha solution multiple times and "replay" is explicitly disabled for the captcha being used.

Behavior: Because captcha's prevent automation, attackers will sometimes try and find ways to abuse the technique used to request the captcha in order to exploit the site. For example, if the attacker can find a way to submit the same solution over and over again, they may be able to solve the captcha once and still automate the resulting workflow. This is sometimes considered legitimate behavior (as would be expected if the user refreshed the browser after submitting a successful captcha), however in many cases, such functionality would make the captcha significantly less effective at preventing automation. In this case, the attacker resubmitted a request that had already been successfully validated through a captcha, and "replay" was explicitly disabled for the captcha. This is not necessarily a malicious incident on its own, because the user may have accidentally refreshed the browser, however multiple attempts would definitely represent malicious intent. An example of where a captcha's "replay" could cause a problem is on a gaming site, where the user is adding fake "money" to their account. In order to add the fake money, they must solve the captcha. This workflow is protected with a captcha, because if a user could automate the process, they would be able to add unlimited funds to their account. If an attacker were able to solve the captcha once, and continuously resubmit the resulting request, they could effectively add funds over and

over again without resolving a new captcha. This would then allow for automation. Replay attackers are less of a problem if the web application being protected already has a method of preventing the same request from being submitted accidentally multiple times. Such would be the case if the web application maintained state information for the given session, and recorded the operation after it was successful, then used that state information to prevent a future occurrence of the operation.

Response Processors: Request Captcha Processor: Incident - Multiple Captcha Disallow Multipart

Complexity: Low (2.0)

Default Response: 1x = 1 Day Clear Inputs

Cause: A CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a special technique used to differentiate between human users, and automated scripts. The user is required to visually identify characters in a jumbled image and transcribe them into a text box. An audio version is also available, for users with a visual handicap. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. CAPTCHAs are used in two different ways by the System. They can be explicitly added to any workflow within the protected web application (such as requiring a CAPTCHA to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). CAPTCHAs are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of CAPTCHA is being used, this incident is generated when the user attempts to submit dozens of CAPTCHA-protected requests that contain binary files, and the CAPTCHAs are explicitly configured to not allow binary file submission (it has been configured to disallow multi-part form submissions).

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the CAPTCHA, they may use various techniques. One of these techniques is to try changing various values used by the web application in the CAPTCHA mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse-engineering is generally performed by advanced hackers. In this specific case, the attacker submitted dozens of binary files in the requests that are being protected. The CAPTCHA in this case has been explicitly configured to not allow Multi-Part form submissions, so this represents unexpected and undesired activity. Using Multi-Part forms, the attacker can more easily accomplish a "Buffer Overflow" attack, which would produce potentially sensitive error data and possibly open the server up to further exploitation. Additionally, some web applications do not handle the encoding used for multi-part forms gracefully, so error information may also be obtained from conflicts arising from the submission type. Because this is happening so frequently from the same user, it is also possible that the user is attempting to execute a "Denial of Service" attack.

Response Processors: Request Captcha Processor: Incident - Multiple Captcha Parameter Manipulation

Complexity: Low (2.0)

Default Response: 1x = Warn User, 2x = 1 Day Clear Inputs

Cause: A CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a special technique used to differentiate between human users, and automated scripts. The user is required to visually identify characters in a jumbled image and transcribe them into a text box. An audio version is also available, for users with a visual handicap. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly-impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. CAPTCHAs are used in two different ways by the System. They can be explicitly added to any workflow within the protected web application (such as requiring a CAPTCHA to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). CAPTCHAs are generally used to resolve "Insufficient Anti-Automation" weaknesses in the protected web application. Regardless of which type of CAPTCHA is being used, this incident is generated when the user attempts to submit multiple solutions for multiple CAPTCHAs, but they keep modifying the query parameters that were submitted with the original requests. For example, if the user submitted a "add product to cart" request, and one of the query parameters was the item to add, this incident would be triggered if, after solving the CAPTCHA, the value of that query parameter was modified to some other value, and this modification happened dozens of times.

Behavior: Because CAPTCHAs prevent automation, attackers will sometimes try to find ways to abuse the technique used to request the CAPTCHA in order to exploit the site. For example, if the attacker can find a way to submit the same solution over and over again, but have the web application perform a different action each time, they may be able to solve the CAPTCHA once, and still automate the resulting workflow. In this case, the attacker changed many query parameters on many different requests that were protected with a CAPTCHA. They submitted the original request, solved the CAPTCHA, changed the original query parameters, and then resubmitted the solved CAPTCHA request. In some cases, this might cause the web application to execute a different operation based on the difference in query parameter values. For example, if the protected workflow is "add product to cart" on a shopping site, then the attacker might attempt to submit the same solved CAPTCHA repeatedly, but change the product ID that is being added on each request. This might allow them to automate the addition of products to a shopping cart, after solving only one CAPTCHA challenge. The CAPTCHA mechanism does not allow the modification of query parameters after the original request has been submitted, so this type of activity will not be successful. This is not considered malicious activity right away, because it is possible that a user may accidentally modify a query parameter; however, when this incident is triggered, it represents a user who has modified dozens of different query parameters on different CAPTCHA-protected pages.

Response Processors: CSRF Processor

The CSRF processor is responsible for ensuring that the protected website does not allow a cross site request forgery attack. CSRF attacks are a type of session hijacking, where a malicious website redirects a user to a sensitive service call on the target website. For example, a user might visit a malicious website that has an image tag pointed to the "deleteAccount" service running on a target website. When a user visits the malicious website, they are unknowingly calling the "deleteAccount" operation. If they had an active session on the target site, their account would be deleted.

This processor works by intercepting any request that could potentially be part of a CSRF attack. This is determined by looking at the referer header being passed in by the client. The referer header tells the server where the user came from. If the user is navigating around the actual website legitimately, they will have a referer header on nearly all requests they make which will match the domain of the site they are navigating. If the user types the URL in manually, or follows a link from another site, they will not have a referer. If it's a CSRF attack, there will either be no referer, or a 3rd party domain in the referer.

In all cases where the referer does not match the domain of the protected site, a special redirection page will be returned to the client instead of the request they actually asked for. The redirection page will check to make sure the user is not a victim of a CSRF attack, and if they are not, it will automatically redirect the user to the original page they requested.

This processor only protects clients that have "user-agent" headers matching that of a known browser. This is because CSRF attacks are specifically targeted at average web users, and they generally stick to the major browsers. So spiders and scripts will bypass the CSRF processors detection/protection mechanism. This processor also detects the case where a user has turned off referers (and thus, no requests will contain a referer), and in that case, will turn off CSRF protection for the client. As such, a user who has disabled referers will still be susceptible to CSRF, but that should be a very small percentage (if not zero) of the overall user pool.

In the event that a user issues a request that cannot be validated as not a CSRF attack, the user will not be automatically redirected. Instead, they will be presented a "This page has moved" response, and will be asked to click a link to continue to the page they actually wanted. The link to proceed is randomly positioned on the page to prevent Click Jacking attacks (where a malicious site overlays legitimate content on top of the target site and gets the user to click the legitimate content, while also hijacking the click to transparently activate the content underneath). A special case involves when a 3rd party website opens the target site in a new window or tab. If the 3rd party site retains ownership of the newly opened window or tab, the user will be asked to click the "continue" link so that the original window can be closed and a new window can be opened in its place. This action breaks the ownership and prevents the 3rd party website from performing actions on the window (such as closing or redirecting it).

Because it is sometimes expected that a 3rd party site will be making calls into the target site, it is possible to configure a list of "trusted" 3rd party sites. Any requests issued from

a trusted domain will not be protected against CSRF. This allows the trusted site to host the target site in an IFRAME or make service calls unimpeded. Be careful who you add to the trusted domain list, because if the trusted domain is susceptible to XSS or CSRF itself, then it can be used as a proxy to launch a CSRF attack against your protected sites. This trust does not apply if the hosting domain is running over SSL, and the target domain is not running over SSL. If the 3rd party page hosting an IFRAME of the target site is running in SSL, it must load the SSL version of the target site, otherwise the CSRF protection will still be applied. It is however fine if the 3rd party site is not SSL protected and the target site is SSL protected.

Table 31: CSRF Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Block Response	Configurable	HTTP Response	The response to return if the CSRF mechanism cannot complete the request due to errors or tampering.
CSRF Nonce Salt	String	Random	A 256 character random string used to ensure that CSRF nonce tokens are generated differently between different deployments.
CSRF Token Name	String	Random	The name of the query string parameter used to indicate a successfully validated request after it has been determined that it is not a CSRF attack. Select a name that will not conflict with a real query parameter used by the site.
Ignore Scripts	Boolean	True	CSRF is largely a browser based attack, so to ensure that scripts such as legitimate spiders are not treated as potential CSRF victims, this option can be enabled to ignore all non browsers for CSRF protection.
Ignored Extensions	Collection	.xap, .xaml	A list of file suffixes (extensions) that will not be protected by CSRF. By default, Silverlight binaries are included, because some browsers will remove the referer for Silverlight embedded content, which may interfere with CSRF protection and prevent the Silverlight content from loading.
Remote Script Resource	Configurable	CSRF Script Inclusion Resource	The fake resource to request if the page is being loaded as a remote script on a third party domain. This is primarily for detection of the attack and can be any fake resources as long as it does not actually exist on the server.
Trusted Domains	Collection	None	The list of domains that are allowed to display the web application in a frame, reference resources such as images or scripts, or are allowed to make remote API calls using techniques that are similar to a CSRF attack. If the trusted domain starts with a period, then it will match any subdomain before the designated period. For example, .site.com will match www.site.com, my.new.site.com and site.com.

Table 31: CSRF Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
CSRF Extra JavaScript	String	None	Since CSRF protection may cause the referer to be removed from the request, it may be necessary to add any analytic code to the JavaScript used to detect and stop CSRF attacks. As such, if you use a 3rd party analytics script, you should put that code in this parameter to capture the unmodified original request details. The code will be injected into a script tag, so it must be valid JavaScript or the CSRF protection may stop functioning correctly.
Incident: CSRF Parameter Tampering	Boolean	True	The user tampered with the parameters used by the security engine to prevent CSRF on requests that have an untrusted 3rd party referer. This is likely in an attempt to find a vulnerability in the CSRF protection mechanism.
Incident: CSRF Remote Script Inclusion	Boolean	False	The user has accessed an untrusted 3rd party website which contains an embedded script reference to the protected application. While the user may not be malicious, this represents a CSRF attack from the untrusted website against the protected application. Because the attack was not successful, it is likely being executed by the user who is attempting to construct the attack vector.
Incident: HTTP Referers Disabled	Boolean	True	The user is using what looks like a browser, but they have HTTP referers disabled. This is not a malicious incident, but it does indicate an unusual client.

Response Processors: CSRF Processor: Incident - CSRF Parameter Tampering

Complexity: Suspicious (1.0)

Default Response: 10x = Multiple CSRF Parameter Tampering Incident.

Cause: WebApp Secure protects against CSRF attacks by using a special interception technique. When a request comes in to WebApp Secure, the referer is checked. In the event that there is a 3rd party referer (the user was following a link from another site), the interception mechanism kicks in. This involves returning a special page to the user that validates that the user is intentionally requesting the resource. If the validation is successful, the user is transparently redirected to the original resource they requested. If the validation fails, the user is then instructed to manually confirm their intentions, or return to the page they came from (to prevent the CSRF attack from working). In most cases, a valid CSRF attack would function in such a way as to hide this manual confirmation step, so the user would probably never see it (e.g. if the URL was loaded using an image HTML tag, then the resulting HTML confirmation step would not render, because its HTML, not an image). This incident is triggered when a user submits a request with a 3rd party referer, and then manipulates the code of the CSRF interception page to alter the original data that was submitted. For example, they submit a request that looks like a CSRF attack (has a 3rd party referer), and then use a tool like Firebug to edit the query string parameters that would be sent to the server after they manually allowed the request on the CSRF intercept page.

Behavior: CSRF attacks are generally two-phase. The first phase involves the attacker establishing a functional CSRF attack. This could take quite a while and involves the attacker making requests to the protected site, trying all different types of CSRF techniques. The second phase is when the attacker injects the successful CSRF vector into a public website. In the second phase, legitimate users are visiting the public website and unknowingly executing the CSRF attack in the background. It is not useful to flag the victims of the CSRF attack as hackers, because they may not even know what is going on. However it is useful to flag the original attack vector establishment, because it may shed light on who created the "CSRF243" attack. This incident reflects a user who is manipulating the CSRF prevention mechanism, likely in an attempt to find a way to get around it. As such, if a user has this incident, they are probably trying to establish a CSRF attack, and careful attention should be paid to the values they are changing the parameters to and which URL is being requested (this will help identify what the user is trying to attack).

Response Processors: CSRF Processor: Incident - Multiple CSRF Parameter Tampering

Complexity: Low (2.0)

Default Response: 1x = Captcha, 2x = 1 Day Clear Inputs

Cause: WebApp Secure protects against CSRF attacks by using a special interception technique. When a request comes in to WebApp Secure, the referer is checked. In the event that there is a 3rd party referer (the user was following a link from another site), the interception mechanism kicks in. This involves returning a special page to the user that validates that the user is intentionally requesting the resource. If the validation is successful, the user is transparently redirected to the original resource they requested. If the validation fails, the user is then instructed to manually confirm their intentions, or return to the page they came from (to prevent the CSRF attack from working). In most cases, a valid CSRF attack would function in such a way as to hide this manual confirmation step, so the user would probably never see it (e.g. if the URL was loaded using an image HTML tag, then the resulting HTML confirmation step would not render, because its HTML, not an image). This incident is triggered when a user submits dozens of requests with a 3rd party referers, and then manipulates the code of the CSRF interception page to alter the original data that was submitted. For example, they submit a bunch of requests that look like CSRF attacks (they have 3rd party referers), and then use a tool like Firebug to edit the query string parameters that would be sent to the server after they manually allowed the requests on the CSRF intercept page.

Behavior: CSRF attacks are generally two-phase. The first phase involves the attacker establishing a functional CSRF attack. This could take quite a while and involves the attacker making requests to the protected site, trying all different types of CSRF techniques. The second phase is when the attacker injects the successful CSRF vector into a public website. In the second phase, legitimate users are visiting the public website and unknowingly executing the CSRF attack in the background. It is not useful to flag the victims of the CSRF attack as hackers, because they may not even know what is going on. However it is useful to flag the original attack vector establishment, because it may shed light on who created the "CSRF" attack. This incident reflects a user who is manipulating the CSRF prevention mechanism, likely in an attempt to find a way to get around it. As such, if a user has this incident, they are probably trying to establish a CSRF

attack, and careful attention should be paid to the values they are changing the parameters to and which URL is being requested (this will help identify what the user is trying to attack).

Response Processors: CSRF Processor: Incident - CSRF Remote Script Inclusion

Complexity: Informational (0.0)

Default Response: None.

Cause: WebApp Secure protects against CSRF attacks by using a special interception technique. When a request comes in to WebApp Secure, the referer is checked. In the event that there is a 3rd party referer (the user was following a link from another site), the interception mechanism kicks in. This involves returning a special page to the user that validates that the user is intentionally requesting the resource. If the validation is successful, the user is transparently redirected to the original resource they requested. If the validation fails, the user is then instructed to manually confirm their intentions, or return to the page they came from (to prevent the CSRF attack from working). In most cases, a valid CSRF attack would function in such a way as to hide this manual confirmation step, so the user would probably never see it (e.g. if the URL was loaded using an image HTML tag, then the resulting HTML confirmation step would not render, because its HTML, not an image). This incident is triggered when a user accesses a page on a 3rd party website which contains a Javascript tag that loads content from the protected site. This would normally represent a victim of a CSRF attack, but because CSRF attacks are blocked, an attacker is unlikely to execute such an attack. Therefore, it is more probable that the attacker is testing a possible vector to see if it will work and encountering this incident.

Behavior: CSRF attacks are generally two-phase. The first phase involves the attacker establishing a functional CSRF attack. This could take quite a while and involves the attacker making requests to the protected site, trying all different types of CSRF techniques. The second phase is when the attacker injects the successful CSRF vector into a public website. In the second phase, legitimate users are visiting the public website and unknowingly executing the CSRF attack in the background. It is not useful to flag the victims of the CSRF attack as hackers, because they may not even know what is going on. However it is useful to flag the original attack vector establishment, because it may shed light on who created the "CSRF" attack. While this incident would potentially be fired for any victims of a CSRF attack, CSRF attacks are blocked by this processor, so it is unlikely that an attacker would ever actually try to use the vector against legitimate users. As such, it is far more likely that the attacker is still in the first phase and trying to uncover a successful CSRF vector. Because of this, careful attention should be paid to the URL that is being requested (this will help identify what the user is trying to exploit).

Response Processors: CSRF Processor: Incident - HTTP Referers Disabled

Complexity: Suspicious (1.0)

Default Response: None.

Cause: The HTTP protocol provides support for a special header called the "referer" (misspelled on purpose). This header tells the web server where the user just came from. So if the user visits google and follows a link from google to get to another page, the request for that second page will contain a "referer" of "http://www.google.com". Some browsers provide the option to turn off automatic transmission of the "referer" header. This would make it impossible for websites to identify the page the user came from. This incident is triggered whenever a user accesses the website with referers disabled. This is not necessarily a malicious act, as it could be the result of an excessively paranoid legitimate user, but it is also somewhat unusual and is often a technique employed by malicious users.

Behavior: Hackers will often disable the referer header to make it more difficult to monitor and analyze an attack through the traditional HTTP log files. Many web servers will record the URL the user is accessing, as well as the referer that was submitted. As such, by disabling referers, the hacker is able to eliminate a large percentage of the information collected about the attack.

Response Processors: Header Injection Processor

This processor provides the header injection counter response. It allows extra a custom header to be defined that is injected into a suspected hackers requests to allow custom handling.



NOTE: There are no actual triggers for this processor; it is a form of response.

Table 32: Header Injection Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Default Header Name	String	Random	The default header name to use if one is not specified in the response configuration.
Default Header Value	String	True	The default header value to use if one is not specified in the response configuration.

Response Processors: Force Logout Processor

This processor provides the force logout counter response. It strips out and invalidates the users session tokens logging them out of the site.



NOTE: There are no actual triggers for this processor - it is a form of response.

Table 33: Force Logout Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Application Session Cookie	Collection	Collection	A collection of names to use for the Application session cookie.
Advanced			
Clear Session Cookies	Boolean	False	Whether to clear any terminated session cookies from the malicious users browser. This may help the user identify why they are getting logged off, so unless the application has code on the client that reads the session cookie value, or the cookie is used in traffic not protected by the WebApp Secure system, this option should be turned off.

Response Processors: Strip Inputs Processor

This processor is used to transparently remove all user input from requests being issued to the server. This response will make the web application, or the client accessing it, to appear broken from the users perspective. The web site will also take on a much smaller attack surface should the client be a vulnerability scanner.



NOTE: There are no actual triggers for this processor; it is a form of response.

Table 34: Strip Inputs Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.

Response Processors: Slow Connection Processor

The slow connection processor is designed to introduce large delays in requests issued by malicious traffic without impacting the performance of legitimate users. There are no actual triggers for this processor; it is a form of response.



NOTE: If default minimum and maximum delay times for the Slow Connection Processor are set to a value greater than the Backend Response Timeout (Configuration > Proxy/Backends > Connection Timeout), the connection may timeout, resulting in a 403 error.

Table 35: Slow Connection Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Default Maximum Delay	Integer	5 Seconds	The default maximum number of milliseconds to delay malicious requests.
Default Minimum Delay	Integer	500 Milliseconds	The default minimum number of milliseconds to delay malicious requests.

Response Processors: Warning Processor

The warning processor is designed to allow a warning message to be presented to a user without completely blocking site access. The warning processor only enables the ability to respond to a user with a "warning", which would allow them to continue browsing the page and the site. The warning would be created and activated for a user by the auto response system, or manually from the console. The existing processor overlays semi-transparent HTML elements on top of the entire webpage, which temporarily disables any mouse or keystrokes on the page and, therefore, creating a "modal dialog" effect. This processor isn't designed to completely stop an attacker from using the website; it is there to warn them. Given the browser debugging tools available today, an attacker may be able to dismiss the warning by means of such tools. Any tampering with the warning's default dismissal behavior (waiting 5 seconds until dismissal button is automatically enabled and clicking on dismiss button) will be considered an incident and will be tracked.

Table 36: Warning Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			

Table 36: Warning Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Default Warning Message	String	"Your connection has been detected performing suspicious activity. Your traffic is now being monitored."	The default message to use in the warning dialog. This can be defined on a session by session basis, but if no explicit value is assigned to the warning, this value will be used.
Default Warning Title	String	Security Warning	The default title to use in the warning dialog. This can be defined on a session by session basis, but if no explicit value is assigned to the warning, this value will be used.
Dismissal Delay	Integer	10 Seconds	The amount of time in seconds that must elapse before the warning can be dismissed. This is a soft limit, as an experienced user may be able to get around enforcement measures.
Dismissal Resource	Configurable	Random	The information needed to define the URL and response used to dismiss a warning.
Warning Directory	String	Random	The name of the directory where the warning Javascript and css code will be served from. For example: warningcode.
Incident: Warning Code Tampering	Boolean	True	The user has attempted to dismiss the warning without waiting the delay and using the provided mechanism. This is probably an attack on the warning system.

Response Processors: Warning Processor: Incident - Warning Code Tampering

Complexity: Medium (3.0)

Default Response: 1x = Logout User, 2x = 5 Day Clear Inputs.

Cause: WebApp Secure is capable of issuing non blocking warning messages to potentially malicious users. These warning messages are designed to force the user to wait for a period of time, before they can dismiss the warning and continue using the site. If the user attempts to exploit or bypass this delay mechanism in order to dismiss the warning early, this incident will be triggered.

Behavior: Once a hacker has been warned, they are then aware that a security system is monitoring their activity. This may cause some hackers to investigate what might be protecting the site. This could involve additional scanning, or it could involve attacking the warning mechanism directly. This type of behavior generally indicates a hacker with moderate to advanced skill levels. Depending on what they modify the warning code input to be, this could represent a simple exploratory test, or the user could be trying to launch a more complex attack against the warning code handler itself, such as "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", and "SQL injection" among many others.

Response Processors: Application Vulnerability Processor

The application vulnerability processor is designed to block known attack vectors for select 3rd party applications. By default this processor does nothing. If you host a 3rd party application such as WordPress, you should enable the configuration parameters that represent the 3rd party software you are using. This will enable protection for that software component.

Table 37: Application Vulnerability Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Joomla Vulnerability Protection Enabled	Boolean	False	Whether traffic should be analyzed for Joomla vulnerabilities
PHPBB Vulnerability Protection Enabled	Boolean	False	Whether traffic should be analyzed for PHPBB vulnerabilities
Wordpress Vulnerability Protection Enabled	Boolean	False	Whether traffic should be analyzed for Wordpress vulnerabilities
Advanced			
Mode of Operation	Integer	1	Whether to block a request on a positive signature, or just create an incident
Block Response	HTTP Response	404 Error	The default message to use in the warning dialog. This can be defined on a session by session basis, but if no explicit value is assigned to the warning, this value will be used.

Response Processors: Application Vulnerability Processor: Incident - App Vulnerability Detected

Complexity: Low (2.0)

Default Response: 1x = Slow Connection 2-6 seconds, 3x = Slow Connection 2-6 seconds and Clear Inputs for 1 day

Cause: The application vulnerability processor is designed to identify known attack vectors issued to 3rd party applications such as WordPress. This incident indicates that one of those known attack vectors has been issued by the associated user. The exact nature of the vector that was identified should be described in the incident details.

Behavior: One of the easiest ways to compromise a web site is to look for 3rd party web applications such as WordPress. If one is found, the attacker can then look up any known vulnerabilities in that software and the version of it that is running on the web site. If they

find vulnerabilities, they can then launch them and potentially compromise the site with a few minutes with minimal effort.

Response Processors: Support Processor

When a user is blocked or otherwise responded to using one of the counter measures, this processor provides a way to identify which profile is associated with a user, and to then allow those responses to be deactivated at the discretion of the IT administrator. For example, if a user were to get a 404 error when asking for a PDF document linked from the main site, and they then try to find the file by trying a bunch of different file names, they may eventually get blocked for performing a directory enumeration attack. When this happens, the blocked user may contact support for assistance getting access to the site again.

This processor works by exposing a special administrative URL (defined in configuration) which the support team can access. When a support request comes in from a blocked users, the support representative can access this administrative URL which will provide another URL. The support representative should then provide this second URL to the affected user. The affected user can then visit that URL and get a special code. This code can be used to search for the profile and deactivate responses in the Security Monitor (profile list).

If the affected user gets a code of "00000000000000000000" (all zeros), this means that the user is not identified as an attacker and therefore is not being blocked or responded to with a counter response from WebApp Secure. As such, other causes of the user's inability to access the site should be investigated.

DO NOT GIVE OUT THE ADMINISTRATIVE URL. It is only used to get a fresh URL that is safe to provide to the affected user. If the administrative URL is leaked to the public, it should be changed immediately.

The overall workflow is as follows:

1. User is blocked or otherwise responded to with a counter measure.
2. User calls support for assistance.
3. Support accesses the administrative URL.
4. Support copies the newly created URL in the response and provides to the affected user.
5. The affected user accesses the newly created URL and provides the resulting code to support.
6. Support or an Admin then logs into the security monitor, clicks on the profile graph to get a list of profiles, and then searches for the code.
7. Support or Admin reviews user's list of incidents to verify the user was responded to in error. If so, the Support or Admin disables the responses.



NOTE: Note that the "block" response is by default, configured to return the code. So if a user has been blocked, steps 3–5 can be omitted, and the user can simply provide the code specified in the block message to support. For all other responses, the full workflow needs to be followed, because there is no other way to obtain the code.

Table 38: Support Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Private Support URL	String	Random	The URL a support representative would access to get additional details about how to provide support to users who are having issues that may be WebApp Secure related. If the value is "ABC", then the private URL would be <code>http://www.example.com/ABC</code> . It is absolutely imperative that this URL not be leaked to non-internal users. If it is leaked, it must be changed immediately.
Public Support URL Salt	String	Random	A random value used to ensure that support URLs are not predictable. This can be any random string 30 characters in length.
Public URL Expiration	Integer	3	The number of days a public support URL remains valid for. After this many days, the URL will no longer provide support information. This is to prevent any issues from a public support URL being leaked.

Response Processors: Cloppy Processor

The Cloppy processor is a joke response built for demonstration purposes. It creates an animated paperclip in the lower right corner of the website, which belittles and taunts the attacker. This should never be used on a legitimate threat and is not the default counter response for any type of behavior. It is provided to demonstrate the diversity of counter responses WebApp Secure is capable of. You should never activate this response unless you have a good relationship with the user you are activating it on, and they have a good sense of humor.

You can configure the message and options cloppy presents both in configuration (the default messages), or in the response specific config (the XML you define when you manually activate a response or when you write a rule that activates a response). The oldest cloppy response will be the one for which the messages are loaded, so if you create multiple cloppy responses, you can create a dialog of several messages. For example, try activating cloppy three times with the following config values (create them in the following order):

1. Activate Cloppy: `<config message="This is the first message"><option label="First op" url="" /><option label="Second op" url="" /></config>`
2. Activate Cloppy: `<config message="This is the second message"><option label="First op" url="" /><option label="Second op" url="" /></config>`
3. Activate Cloppy: `<config message="This is the third message"><option label="First op" url="" /><option label="Second op" url="" /></config>`

Once you activate the above 3 cloppy responses, you should see that cloppy will present the "This is the first message" dialog first. Once you click on an option in that dialog, the next page you load will display "This is the second message", and finally, after clicking on one of those options, you should get "This is the third message".

Once you click an option in the cloppy's dialog, it will dismiss that specific cloppy response. That's why you are able to stack the responses and get a dialog going.

Table 39: Cloppy Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor. Note that just because traffic is passing through the processor, does not mean any users will actually have a Cloppy response activated on them. As such, simply enabling this processor will not result in cloppy being activated for any users. You would still need to manually activate the Cloppy response in the Security Monitor (or define an auto response rule that activates it, but that is highly discouraged).
Cloppy Message	String	"It looks like youre an unsophisticated script kiddie attempting to hack this web site"	What do you want cloppy to say when offering help?
Cloppy Options	Collection	Collection	The list of ways cloppy can help with associated URLs.
Advanced			
Cloppy Directory	String	cloppybin	The name of the directory where the binary resources needed to load cloppy are served from. For example: cloppyfiles. The name should be selected not to conflict with a real directory at the top level of the web site.
Cloppy Dismiss Directory	String	Random	The name of the directory used to dismiss cloppy. This URL should be random and not conflict with existing directory names on the site.

Response Processors: Login Processor

The login processor is designed to add additional protection to the login dialogs throughout the protected site. By default, it will not provide any additional protection, and must be

configured to protect specific login forms. Once a login form has been configured, the processor will begin to monitor the login attempts and start checking for abusive patterns.

This processor is capable of detecting a wide variety of abuse patterns on a login dialog, as well as stopping these abusive activities. One key protection mechanism is to require a captcha if a user attempts to login to an account which has experienced more than 3 failed login attempts since the last successful login attempt. This ensures that a malicious user cannot brute force a specific username, because after 3 failed attempts, the brute force tool will be stopped by a captcha. This does not represent a counter response, but instead is built in functionality that applies to all users on the system. So if user "A" submits 3 bad passwords, and then user "B" submits a password for the same username, user "B" will get a captcha, as well as user "A" for any additional login attempts they try. As soon as a user successfully logs into the account, it will take another 3 failed login attempts before the next captcha is required.

In addition to protecting against a single username being attacked with a brute force script, the processor also detects "User sharing", "User pooling", "Username scans", "Multi-User brute force scans". See the incident descriptions for more information on what these incidents represent and what counter responses will be activated as a result. In order to configure the Login Processor to protect a login form, edit the "Protected Login Pages" configuration parameter. Add a new row and provide the following information. It will be useful to look at the HTML source code of the login form as it will have critical information you will need to configure protection:

- **Name:** The name of the login page (this is just for your reference, it can be anything)
- **URL Pattern:** The Regular Expression used to identify a username/password submission. This pattern should match the "action" attribute of the HTML <FORM> tag wrapping the login dialog.
- **Username Field Type:** The type of inputs used to submit a username. Normally this will be "POST Parameter", however other options are provided for more specialized login mechanisms.
- **Username Field Name Pattern:** A regular expression used to match the name of the input the username is submitted with. Normally this is "username", but could be other variations such as "usr", "user", etc. You can simply enter the name of the input in this field if a regular expression is not required.
- **Username Field Pattern Value:** A regular expression used to extract the username from the input value. Normally this should just be "^.*\$", but if the username is wrapped in JSON for example, you may need to create a more complex expression. The username is considered the first matching parenthesis group in the pattern.
- **Username Field Encoding:** The type of data encoding used on the username. Normally this will be "Ascii", however if any client side encoding is performed, other encoding options are available.
- **Password Field Type:** The type of inputs used to submit a password. Normally this will be "POST Parameter", however other options are provided for more specialized login mechanisms.

- **Password Field Name Pattern:** A regular expression used to match the name of the input the password is submitted with. Normally this is "password", but could be other variations such as "pwd", "pass", etc. You can simply enter the name of the input in this field if a regular expression is not required.
- **Username Field Pattern Value:** A regular expression used to extract the password from the input value. Normally this should just be "^.*\$", but if the password is wrapped in JSON for example, you may need to create a more complex expression. The password is considered the first matching parenthesis group in the pattern.
- **Password Field Encoding:** The type of data encoding used on the password. Normally this will be "Ascii", however if any client side encoding is performed, other encoding options are available.
- **Failure Pattern Target:** In order to identify a failed login attempt, the processor will search for a specific pattern in the response. This attribute specifies where to search for that pattern. Normally this would be "Body" to search the HTML body of the response.
- **Failure Pattern:** The regular expression to search for to check and see if the login attempt was unsuccessful. Assuming the Failure Pattern Target is "Body", this would be something like "you have provided an invalid username and password". However the exact text will need to be set to whatever the site actually returns. View the source of the response after a failed login and search for the error text, so that you get the most accurate version possible. Simply copying the text from the rendered page may exclude embedded HTML tags which will cause the pattern to never match.
- **Failure Pattern Condition:** Specifies whether finding the failure pattern means the login was unsuccessful, or whether not finding the pattern means the login was unsuccessful.
- **Success Pattern Target:** In order to identify a successful login attempt, the processor will search for a specific pattern in the response. This attribute specifies where to search for that pattern. Normally this would be "Body" to search the HTML body of the response.
- **Success Pattern:** The regular expression to search for to check and see if the login attempt was successful. Assuming the Success Pattern Target is "Body", this would be something like "you have successfully logged in". However the exact text will need to be set to whatever the site actually returns. View the source of the response after a successful login and search for something that only gets displayed on a successful login, so that you get the most accurate version possible. Simply copying the text from the rendered page may exclude embedded HTML tags which will cause the pattern to never match.
- **Success Pattern Condition:** Specifies whether finding the success pattern means the login was successful, or whether not finding the pattern means the login was successful.
- **Require Captcha After:** Specifies how many failed login attempts on the same username before requiring all future login attempts on that username to solve a captcha. Entering "0" will allow infinite attempts.

Keep in mind that some website implementations allow login information to be posted to many different URLs. If that is the case, make sure the URL pattern is defined generically

enough to match any URL the user might submit a login request to. Only submissions that match the URL pattern will be protected.

Once a login form has been configured, it can be tested by attempting to login to the same username 6 or more times. You should be presented with a captcha. Next, solve the captcha and log in with the correct password. Then logout and attempt to login to the same username again. If you do not get a captcha, then the login form is configured correctly.

Table 40: Login Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor. Note that just because this processor is enabled, does not mean that any login forms are being protected. Login forms will not be protected until they are configured in the "Protected Login Pages" parameter.
Protected Login Pages	Collection	None	The list of pages that should be protected from login and account abuse. These pages should reflect the URL's that accept username's and passwords and allow login, not necessarily the pages that contain login forms. For example, if every page on the site had a login form, but they all submitted to login.php, then only login.php needs to be configured in this processor.
Advanced			
Bad Request Block Response	HTTP Response	400 Error	The response to return if the user issues a request that either is too large, or uses multipart and multi-part is disabled.
Blocked Replay Response	HTTP Response	400 Error	The response to return if the user attempts to submit the validated request multiple times using the same captcha answer, and that behavior is not allowed.

Table 40: Login Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Cancel URL	String	(empty)	The URL to redirect the user to if they cancel the captcha. This should not be to the same domain, because the domain is being blocked using a captcha, and therefore, canceling would only redirect to a new captcha. An empty value will hide the cancel button.
Captcha Binary Directory	String	Random	The name of the directory where captcha images and audio files will be served from. This should not conflict with any actual directories on the site.
Captcha Characters	String	abcdefghijklmnopqrstuvwxyz ABCEFGHJKLMNPQRTWXYZ 234678	The characters to use when generating a random captcha value. Avoid using characters that can be easily mixed up. This set of characters is case sensitive.
Captcha Expiration	Integer	120	The maximum number of seconds the user has to solve the captcha before the request is no longer possible.
Captcha State Cookie	String	Random	The name of the cookie to use to track the active captchas that have not yet been solved. The cookie is only served to the captcha binary directory.
Captcha Template	File	Default Template	The HTML template used to ask the user to complete a captcha. This template must contain specific key words in order to integrate properly. Please refer to the manual for more information.
Captcha Validation Input Name	String	Random	The name of the form input used to transmit the captcha validation key. This should be obscure so that users who have not been required to enter a captcha cannot supply bad values to this input to profile the system.

Table 40: Login Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Expired Captcha Response	HTTP Response	400 Error	The response to return if the user submits a validated request after the captcha has expired. This may happen if the user refreshes the results of the captcha long after they have solved it.
Maximum Active Captchas	Integer	7	The maximum number of captchas any given user can be solving at any given time. This limit can be overcome, but the majority of users will not be able to. This is primarily for performance, as the more active captchas that are allowed, the larger the state cookie becomes.
Maximum Request Size	Integer	524288 (500KB)	The maximum number of bytes in a request before it is considered not acceptable for captcha validation, and will be blocked.
Support Audio Version	Boolean	True	Whether an audio version of the captcha is provided to the user. This may be a requirement for accessibility, as vision impaired users would otherwise be unable to solve the captcha.
Watermark	String	%DOMAIN	The text to watermark the captcha with. This can be used to prevent the captcha from being used in a phishing attack. For example, an abuser would not be able to simply display the captcha on a different site and ask a user to solve it. The watermark would tip the user off that the captcha was not intended for the site they are visiting. Use %DOMAIN to use the domain name as the watermark.

Table 40: Login Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Incident: Site Invalid Login	Boolean	True	The user has submitted an invalid username or password. This is just an informational incident and is used to identify more complex attacks. It is highly recommended that this incident not be disabled, as it may cause other incidents to no longer register.
Incident: Site Login Multiple IP	Boolean	True	The user has submitted a valid username and password for an account that has recently been used by a different IP. This is just an informational incident and is used to identify more complex attacks. It is highly recommended that this incident not be disabled, as it may cause other incidents to no longer register.
Incident: Site Login Multiple Usernames	Boolean	True	The user has submitted a valid username and password for more than one account recently. This is just an informational incident and is used to identify more complex attacks. It is highly recommended that this incident not be disabled, as it may cause other incidents to no longer register.

Response Processors: Login Processor: Incident - Site Invalid Login

Complexity: Suspicious (1.0)

Default Response: 15x (3 or more bad passwords per username) = Site Login Brute Force, 15x (less than 3 bad passwords per username) = Site Login Username Scan, 8x (bad passwords for same username) = Site Login User Brute Force.

Cause: The login processor is designed to protect the login dialog of the website. It works by monitoring all login attempts and identifying suspicious and malicious events. This specific incident is triggered when a user attempts to login with an invalid username and password. This incident alone is not considered malicious, but is used to perform additional analysis and may be promoted to a malicious incident if an abusive pattern is identified (such as many invalid logins representing a brute force attack).

Behavior: This incident simply reflects the case where a user has entered bad login information. By itself, this cannot be considered malicious as it is extremely common for a legitimate user to accidentally type their information incorrectly, or to forget their password. As such, it is only an indication of possible abuse and requires additional analysis and data before it can be confirmed as malicious or acceptable.

Response Processors: Login Processor: Incident - Site Login Multiple IP

Complexity: Informational (0.0)

Default Response: 3x = Site Login User Sharing

Cause: The login processor is designed to protect the login dialog of the website. It works by monitoring all login attempts and identifying suspicious and malicious events. This specific incident is triggered when multiple clients successfully log into the same account. Depending on the nature of the protected site, this may be perfectly acceptable behavior, however on some sites this type of behavior can indicate abuse. This incident alone is not considered malicious, but is used to perform additional analysis and potentially promote the event as a malicious incident if an abusive pattern is identified.

Behavior: Many web sites provide a way for users to authenticate so that their experience and data can be customized specifically for them. In the case of this incident, credentials for one of those accounts have been distributed to multiple clients and two or more of those clients are logging into the account. Unless the web site expects users to share credentials, this would generally indicate a situation where the credentials for an account have been compromised and the account has been hijacked. Additional follow up may be required to recover the account (such as changing the password or locking the account until the actual owner contacts the administrators to resolve the issue).

Response Processors: Login Processor: Incident - Site Login Multiple Usernames

Complexity: Suspicious (1.0)

Default Response: 3x = Site Login User Pooling

Cause: The login processor is designed to protect the login dialog of the website. It works by monitoring all login attempts and identifying suspicious and malicious events. This specific incident is triggered when a single client successfully authenticates with multiple distinct usernames. This incident alone is not considered malicious, but is used to perform additional analysis and potentially promote the event as a malicious incident if an abusive pattern is identified.

Behavior: There are two possibilities for this incident. Firstly, a single user may have signed up for multiple accounts on the protected site, and they are simply using those accounts. On some sites, this alone would be considered malicious, while on other sites, this is considered perfectly acceptable. For example, an online email provider may allow its users to sign up for multiple email accounts. On the other hand, a billing web site for your home utility provider would probably not expect a single household to have multiple accounts. The other possibility is that a single user has hijacked several other accounts. This may be more obvious if there is also a "Site Login User Sharing" incident for the username as well. This would indicate that not only is the malicious user logging into

multiple accounts, but other users are also logging into those accounts. Generally, an account should be used by a single user unless the web site has specific rules about allowing users to share account details.

Response Processors: Login Processor: Incident - Site Login User Sharing

Complexity: Low (2.0)

Default Response: None.

Cause: The login processor is designed to protect the login dialog of the website. It works by monitoring all login attempts and identifying suspicious and malicious events. This specific incident is triggered when multiple clients successfully log into the same account. Depending on the nature of the protected site, this may be perfectly acceptable behavior, however on some sites this type of behavior can indicate abuse.

Behavior: Many web sites provide a way for users to authenticate so that their experience and data can be customized specifically for them. In the case of this incident, credentials for one of those accounts have been distributed to multiple clients and two or more of those clients are logging into the account. Unless the web site expects users to share credentials, this would generally indicate a situation where the credentials for an account have been compromised and the account has been hijacked. Additional follow up may be required to recover the account (such as changing the password or locking the account until the actual owner contacts the administrators to resolve the issue).

Response Processors: Login Processor: Incident - Site Login User Pooling

Complexity: Low (2.0)

Default Response: None.

Cause: The login processor is designed to protect the login dialog of the website. It works by monitoring all login attempts and identifying suspicious and malicious events. This specific incident is triggered when a single client successfully logs into several different accounts. Depending on the nature of the protected site, this may be perfectly acceptable behavior, however on some sites this type of behavior can be harmful.

Behavior: There are two possibilities for this incident. Firstly, a single user may have signed up for multiple accounts on the protected site, and they are simply using those accounts. On some sites, this alone would be considered malicious, while on other sites, this is considered perfectly acceptable. For example, an online email provider may allow its users to sign up for multiple email accounts. On the other hand, a billing web site for your home utility provider would probably not expect a single household to have multiple accounts. The other possibility is that a single user has hijacked several other accounts. This may be more obvious if there is also a "Site Login User Sharing" incident for the username as well. This would indicate that not only is the malicious user logging into multiple accounts, but other users are also logging into those accounts. Generally, an account should be used by a single user unless the web site has specific rules about allowing users to share account details.

Response Processors: Login Processor: Incident - Site Login User Brute Force

Complexity: Medium (3.0)

Default Response: 1x = Break Authentication for 1 hour, 2x = Break Authentication for 6 hours, 3x = Clear Inputs for 1 day

Cause: The login processor is designed to protect the login dialog of the website. It works by monitoring all login attempts and identifying suspicious and malicious events. This specific incident is triggered when a user attempts to login to the same username many times with invalid passwords.

Behavior: In this case, the user is probably attempting to brute force the account indicated in the incident details. Brute force against authentication works by enumerating over a list of common passwords and testing all of them against the target username. The hope is that the target user selected a weak password and that password is in the "dictionary" list of passwords to try. In some cases, a custom brute force tool may be employed, which enumerates over a list of passwords that were carefully constructed using the targets personal information (birthdays, anniversaries, names, ages, phone numbers, etc.)

Response Processors: Login Processor: Incident - Site Login Brute Force

Complexity: Medium (3.0)

Default Response: 1x = Slow Connection for 6 hours, 3x = Slow Connection & Break Authentication for 6 hours

Cause: The login processor is designed to protect the login dialog of the website. It works by monitoring all login attempts and identifying suspicious and malicious events. This specific incident is triggered when a user attempts to login against a large number of different usernames.

Behavior: A common authentication attack is Brute Force. This attack involves submitting a large number of username and password combinations in an effort to identify users who have chosen weak passwords. This type of attack is extremely noisy and requires thousands of requests to execute.

Response Processors: Login Processor: Incident - Site Login Username Scan

Complexity: Medium (3.0)

Default Response: 1x = Captcha for 6 hours, 3x = Clear Inputs & Slow Connection for 1 day

Cause: The login processor is designed to protect the login dialog of the website. It works by monitoring all login attempts and identifying suspicious and malicious events. This specific incident is triggered when a user attempts to login against a large number of different usernames with a small number of passwords for each.

Behavior: One flaw present in a lot of authentication implementations is that the results that are returned when submitting an invalid username and password are different than the results returned when the username is valid but the password is not. By enumerating over a large number of possible usernames and supplying bad passwords, the attacker is able to identify which usernames are actually valid in the system. This is one of the first steps to a large scale brute force attack. Once the user has a list of valid usernames, they can then launch the brute force attack against just those usernames to make the attack quicker and harder to identify. A best practice when developing authentication systems is to ensure that the results that are returned from an invalid username, are the same results returned when providing a valid username and invalid password. For example, the error should read "The username and password you supplied could not be found in our database", instead of "The username you provided does not exist".

Response Processors: Google Map Processor

The Google Map Processor provides a counter response called the "Google Map Response". When this response is activated, the user will be shown an overlay dialog with a google map of their geo location (as resolved from their IP address using MaxMind Geo IP). It will then recommend 4 google search results on a configured term (default is 'Criminal Attorney'). The intention is to scare the individual into believing that we know where they live and plan to attempt prosecution.

The google map response requires several things in order to work. First, you must obtain a google map API key and set it in configuration. Until you do this, you will not be able to enable the processor. Once enabled, if you activate the processor on a user, they will only see the response if WebApp Secure can resolve their geo location from MaxMind GeoIP. If a geo location cannot be resolved, the map will not be displayed. Additionally, the google map response is not a default response for any activity, so unless you manually activate it, or create a custom auto response rule to activate it, it will never be used.

Keep in mind that by activating this response, you are effectively broadcasting your public google map API key to the attacker. If the attacker decides to exploit this fact, they can easily drain your google map request and search result quotas. As such, it is important to get an API key for a junk google development account, so that your quota's are not shared with legitimate site functions. You should also not sign up for paid quota extensions on that particular account, as that could allow the attacker to run up your bill. Just use the free quotas.

Table 41: Google Map Processor Configuration Parameters

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	False	Whether traffic should be passed through this processor.
Google API Key	String	[Not Set]	The API key issued by Google to authorize the map API to be used on the domain being protected by WebApp Secure. This API key should be enabled for both Google Map API v3, and the Custom Search API.

Table 41: Google Map Processor Configuration Parameters (*continued*)

Parameter	Type	Default Value	Description
Advanced			
Default Search Term	String	"Criminal Attorney"	The default term to search for localized locations on.
Dismissal Resource	Map Dismissal Resource	mapdata	The information needed to define the URL and response used to dismiss a map.
Map Directory	String	mapdata	The name of the directory where the map Javascript and css code will be served from. For example: mapdata.

CHAPTER 33

Incident Methods

- [List Of Incident Methods on page 267](#)

List Of Incident Methods



NOTE: Parameters wrapped in [] are optional.

Table 42: Incident Methods

Name and Description	Parameters
isIncidentType Check the incident type by either its code or its name.	incident:string
isIncidentDate Check to see if an incident occurred on the given month, day and year. The month, day and year arguments may be left empty to match any value. Note that Jan = 1, and years are in the format YYYY.	[month:int] [day:int] [year:int]
isIncidentDateRange Check to see if an incident occurred between two dates. All values must be defined. Note that Jan = 1, and years are in the format YYYY.	start_month:int start_day:int start_year:int end_month:int end_day:int end_year:int
isIncidentTime Check to see if an incident occurred at a given time. The hour, minute and second arguments may be left empty to match any value.	[hour:int] [minute:int] [second:int]

Table 42: Incident Methods (*continued*)

Name and Description	Parameters
isIncidentTimeRange Check to see if an incident occurred between a given time range. All values must be specified.	start_hour:int start_minute:int start_second:int end_hour:int end_minute:int end_second:int
isIncidentCount Check the number of times an incident has occurred against an integer operation and specified value. Supported operations include (>, <, ==, !=). The results are: (count [operator] value)	operator:string value:int
isIncidentCountRange Check to see if the number of times an incident has occurred is within a given range.	min:int max:int
isIncidentContextSubString Check to see if the context XML associated with the incident contains the provided substring. The search is case sensitive by default, unless the second parameter is "false".	search:string [[caseSensitive]:Boolean]
isIncidentContextPattern Check to see if the context XML associated with an incident contains a simple pattern. Supported pattern wild cards include +, ? and *. Pattern matches are performed case sensitive unless the second parameter to this method is "false".	pattern:string [[caseSensitive]:Boolean]
isIncidentIP Check to see if an incident came from a given IP address. Each parameter specifies the required value for the specific block of the address. Any of the parameters can be left empty to match any value.	[a_block:int] [b_block:int] [c_block:int] [d_block:int]
isIncidentIPRange Check to see if an incident came from a given IP address range. Each parameter specifies a range of accepted values for the specific address block. Ranges are specified in the format: min-max. For example: 10-22, or 0-255	[a_block_range:string] [b_block_range:string] [c_block_range:string] [d_block_range:string]
isIncidentBrowser Check to see if the incident occurred from a given browser. The parameter expects the canonical name of the browser.	name:string
isIncidentOperatingSystem Check to see if the incident occurred from a given operation system. The parameter expects the canonical name of the operating system.	name:string

Table 42: Incident Methods (*continued*)

Name and Description	Parameters
isIncidentBrowserVersion Check to see if the incident occurred from a specified version of the browser. The check is case sensitive by default, unless the second parameter is "false". The version could contain any character and should be considered as an arbitrary user supplied string value.	version:string [[caseSensitive]:Boolean]
isIncidentBrowserVersionPattern Check to see if the incident occurred from a browser with a version that matches a given simple pattern. Pattern wild cards >include ?, * and +. The match is done case sensitive unless the second parameter is "false". The version could contain any character and should be considered as an arbitrary user supplied string value.	pattern:string [[caseSensitive]:Boolean]
isIncidentBrowserVersionSubString Check to see if the incident occurred from a browser with a version that contains the given sub string. The match is done case sensitive unless the second parameter is "false". The version could contain any character and should be considered as an arbitrary user supplied string value.	Search:string [[caseSensitive]:Boolean]
isIncidentCountry Check to see if the incident originated from a given country. The parameter expects a valid 2 character country code, or the canonical name of the country.	country:string
isIncidentLatitude Check to see if the incident originated from a specified geographical latitude. The parameter is expected to be a decimal number between -90.0 and +90.0.	latitude:float
isIncidentLatitudeRange Check to see if the incident originated between a specified geographical latitude range. The parameters are expected to be decimal numbers between -90.0 and +90.0.	min:float max:float
isIncidentLongitude Check to see if the incident originated from a specified geographical longitude. The parameter is expected to be a decimal number between -90.0 and +90.0.	longitude:float
isIncidentLongitudeRange Check to see if the incident originated between a specified geographical longitude. The parameters are expected to be decimal numbers between -90.0 and +90.0.	min:float max:float
isIncidentCity Check to see if the incident originated in a specified city. The parameter is expected to be the city name and is case sensitive unless the second parameter is "false".	city:string [caseSensitive]:Boolean
isIncidentCityPattern Check to see if the incident originated from a city that matches a specified pattern. The supported wild cards are *, ?, and +. The pattern is case sensitive unless the second parameter is "false".	pattern:string [caseSensitive]:Boolean
isIncidentCitySubString Check to see if the incident originated from a city that contains a specified sub string. The substring search is done case sensitive unless the second parameter is "false".	search:string [caseSensitive]:Boolean
isIncidentHost Check to see if the incident originated in a specified host. The parameter is expected to be the host name and is case sensitive unless the second parameter is "false".	host:string [caseSensitive]:Boolean
isIncidentHostPattern Check to see if the incident originated from a host name that matches a specified pattern. The supported wild cards are *, ?, and +. The pattern is case sensitive unless the second parameter is "false".	pattern:string [caseSensitive]:Boolean

Table 42: Incident Methods (*continued*)

Name and Description	Parameters
isIncidentHostSubString Check to see if the incident originated from a host name that contains a specified sub string. The substring search is done case sensitive unless the second parameter is "false".	search:string [caseSensitive]:Boolean
isIncidentRegion Check to see if the incident originated in a specified region. The parameter is expected to be the region name and is case sensitive unless the second parameter is "false".	region:string [caseSensitive]:Boolean
isIncidentRegionPattern Check to see if the incident originated from a region that matches a specified pattern. The supported wild cards are *, ?, and +. The pattern is case sensitive unless the second parameter is "false".	pattern:string [caseSensitive]:Boolean
isIncidentRegionSubString Check to see if the incident originated from a region that contains a specified sub string. The substring search is done case sensitive unless the second parameter is "false".	search:string [caseSensitive]:Boolean
isIncidentZip Check to see if the incident originated in a specified zip code. The parameter is expected to be the zip code and is case sensitive unless the second parameter is "false". While zip codes should generally be numeric, there is the possibility of foreign zip codes containing strange characters.	zip:string [caseSensitive]:Boolean
isIncidentZipPattern Check to see if the incident originated from a zip code that matches a specified pattern. The supported wild cards are *, ?, and +. The pattern is case sensitive unless the second parameter is "false".	pattern:string [caseSensitive]:Boolean
isIncidentZipSubString Check to see if the incident originated from a zip code that contains a specified sub string. The substring search is done case sensitive unless the second parameter is "false".	search:string [caseSensitive]:Boolean

Captcha Template

- [Captcha Template on page 271](#)

Captcha Template

There are several processors that utilize captchas to prevent automation. These processors include:

- Request Captcha Processor

This processor allows you to attach a captcha to any page on the web application. It is also responsible for enforcing the "Force Captcha Validation" counter response.

- Login Processor

This processor utilizes captchas to prevent brute force attacks on login dialogs. Once there have been more than three (3) failed login attempts on a single username (from any users), any future attempts to login as that user will require a captcha.

When a captcha must be presented, the format in which it is displayed is defined as a Captcha Template. By default, there is a captcha template defined for both processors that will work on all websites. In the event that you would like to customize the way the captcha looks when it is presented (such as wrapping it with the standard template of the website being protected), the captcha template can be modified. This is done by accessing the advanced configuration parameters for the two aforementioned processors and editing the "Captcha Template" parameter.

In order to edit the parameter, it is recommended that you first download a copy of the existing default template. If you have already made modifications to the template, you can get the original by selecting the suggestion "Default Unbranded Template", and then downloading the associated file.

Once you have a copy of the default template, open it in a text editor. You can make any modifications to the HTML as required, but be sure not to modify the existing JavaScript or remove any of the existing HTML. To prevent introducing changes that might prevent the captcha from functioning, it is recommended that modifications be limited to stylistic changes (do not alter the content of the SCRIPT tags, and do not alter the contents of the FORM tag). After your modifications, you can upload the new file into the parameter to update the captcha HTML served by WebApp Secure. It is recommended that you keep a copy of the modified template to make future modifications easier.

You will also notice that there are a few special HTML tags in the template. These tags are replaced by WebApp Secure before the template is served to the end-user. These tags reside either in a SCRIPT tag or in a FORM tag, so as long as those elements are not modified, these tags should continue to function correctly. These special tags include:

- **<%captchaDir>** The directory name that all captcha images and audio files are served from.
- **<%signature>** The file name for the captcha image or audio resource to load.
- **<%includeAudio>...<%includeAudio>** Displays the content between the open and closing tags only if audio captchas are enabled.
- **<%cancel>** The URL to redirect the user to if they cancel the captcha operation.
- **<%delay>** The number of seconds the user has to complete the captcha before it expires.
- **<%multiPart>...<%multiPart>** Displays the content between the opening and closing tag only if the original request that is being protected by a captcha was a multipart form submission (vs. a URL encoded form post [by default, forms are URL encoded]).
- **<%datasignature>** The signature of the data that was originally posted to the page protected by the captcha. This is used to ensure that the data is not modified after submission, but before the captcha is solved.
- **<%data>** The encrypted data submitted to the original page that required a captcha. This is used so that once the captcha is solved, the original request can be reconstructed and submitted to the backend servers.
- **<%inputname>** The name of the input used to identify when a user submits a captcha. The value for this input name is configurable and should not conflict with any existing inputs the site uses. A random string of 5 or more characters should be sufficient (but must be set in configuration so that it can be injected in place of the custom tag when serving a captcha).

After the new template has been uploaded and saved in configuration, you can test your changes by triggering the applicable captcha.

- **Request Captcha Processor** Access the protected page and request `http://www.domain.com/.htaccess` which will generate a profile for your session. Find the new profile in the security monitor and manually activate the "Force Captcha Validation" response. Then go back to the protected site and make a few more requests until the captcha shows up.
- **Login Processor** If the login processor is configured to protect a login dialog on the site, then simply provide 3 or more invalid passwords for the same username. On the 4th attempt, you should be presented with the login processor captcha.



.....

NOTE: Note: Changes to the captcha template are made to the live deployment. So if you break the captcha template during modifications, it may cause the captcha to stop working for some of the users on the site until the template is repaired. Creating a new "Page" in configuration for a fictitious URL and making the changes on that page first would allow you to test the modifications without impacting every use on the site.

.....

CHAPTER 35

Log Format

- [Log Format on page 275](#)
- [Incident Log Format on page 275](#)
- [Counter Response Log Format on page 276](#)
- [Profile Log Format on page 277](#)

Log Format

Webapp Secure is configured to log security incidents to **mws-security-alert.log**. The creation of new profiles, incoming incidents, and sent counter responses all have alerts that can individually be turned on or off. The following section explains the format of these security log messages.

All security alert log messages have this common format:

`<date> <host> [<log level>] [mws-security-alert] [<thread ID>] <message>`

Where all of the items in angle brackets will be replaced by information relevant to that log entry. The `<message>` portion of each log entry consists of a series of name value pairs, with the name (unquoted) followed by an equal sign (=) followed by the value (in quotes).

Incident Log Format

If incident logging is enabled, all incidents at or above the configured incident severity level will be sent to syslog in **mws-security-alert.log**.

- **MKS_Category** - will always have the value "Security Incident" when logging security incidents
- **MKS_Type** - a textual name for the type of incident
- **MKS_Severity** - an integer between 0 and 4 for the severity of the incident (0 being lowest, 4 being highest)
- **MKS_ProfileName** - the name of the hacker profile who caused the incident (also visible in the security monitor)
- **MKS_SrcIP** - the ip of the hacker who caused the incident

- MKS_pubkey - a textual key unique to that hacker profile (also visible in the security monitor)
- MKS_useragent - the full useragent string of the browser or other program used by the hacker
- MKS_url - the url used on the request that caused the incident
- MKS_count - the number of times this hacker has caused this same incident

Following the common names will be any incident specific contextual values which are tracked with the incident. These will vary based on incident type. For example a Query Parameter Manipulation incident would include the parameter that was changed along with actual and expected values. Here is a sample log entry:

```
Apr 6 20:58:36 vm1 [INFO][mws-security-alert][Thread-49927] MKS_Category="Security Incident" MKS_Type="Query Parameter Manipulation" MKS_Severity="2"
MKS_ProfileName="Luis 9605" MKS_SrcIP="10.10.10.130"
MKS_pubkey="fkrvpvFNhwoWRgaQiUxS" MKS_useragent="Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.142 Safari/535.19" MKS_url="http://www2.testsite.com:80/basket/?action=listing id=3"
MKS_count="1" MKS_actual="2568" MKS_expected="25304" MKS_param="n_idx"
```

Counter Response Log Format

If counter response logging is enabled, all responses (both autoresponses and manually activated responses) will be logged to syslog.

- MKS_Category - will always have the value "New Counter Response" when logging counter responses
- MKS_ResponseCode - the two letter representation of the counter response issued; for example, "BL" for Blocking response.
- MKS_ResponseName - The full textual representation of the counter response issued; for example, "Block".
- MKS_ProfileId - the id of the profile that the counter response was issued on. This is guaranteed to be unique per attacker, and is used to display the attacker's page within the web UI (<http://HOSTNAME/attackers/PROFILEID>).
- MKS_ProfileName - the name of the hacker profile that the counter response was issued on. (also visible in the security monitor)
- MKS_ResponseCreated - the date and time the response was created.
- MKS_ResponseDelayed - the date and time the response is set to be delayed until.
- MKS_ResponseExpires - the date and time the response is set to expire.
- MKS_ResponseConfig - the configuration used in this counter response.

Here is a sample log entry:

```
Feb 15 15:03:28 vm1 [INFO][mws-security-alert][Thread-214] MKS_Category="New Counter Response" MKS_ResponseCode="CP" MKS_ResponseName="Force Captcha Validation"
MKS_ProfileId="127780" MKS_ProfileName="Peter 4703"
```

```
MKS_ResponseCreated="2013-02-15 15:03:26.576" MKS_ResponseDelayed="2013-02-15
15:03:26.576" MKS_ResponseExpires="null" MKS_ResponseConfig="<config
ix0ix5002='1' />"
```

Profile Log Format

If profile logging is enabled, any new profile creations will be logged.

- MKS_Category - will always have the value "New Profile" when logging profiles.
- MKS_ProfileId - the id of the profile that was created.
- MKS_ProfileName - the name of the profile that was created.
- MKS_PubKey - the public key of the profile that was created, used for unblocking the profile via the support processor.
- MKS_GlobalName - the global name given to the profile from Spotlight service. This entry will only exist if it is a spotlight profile, otherwise it will be omitted.

Here is a sample log entry:

```
Feb 15 15:03:24 zach-vm [INFO][mws-security-alert][Thread-202] MKS_Category="New
Profile" MKS_ProfileId="127780" MKS_ProfileName="Peter 4703"
MKS_PubKey="XAgpFNhjYrQvlaozu2Gb"
```


PART 5

Index

- [Index on page 281](#)

Index

Symbols

#, comments in configuration statements.....	xxi
(), in syntax descriptions.....	xxi
< >, in syntax descriptions.....	xx
[], in configuration statements.....	xxi
{ }, in configuration statements.....	xxi
(pipe), in syntax descriptions.....	xxi

file processor.....	170
---------------------	-----

A

activity processors.....	182
application cookie manipulation.....	187
auth cookie tampering.....	184
auth input parameter tampering.....	183
auth invalid login.....	185
auth query parameter tampering.....	184
authentication brute force.....	185
common directory enumeration.....	195
cookie protection processor.....	186
duplicate request header.....	198
duplicate response header.....	199
error processor.....	187
header processor.....	197
illegal method requested.....	204
illegal request header.....	199
illegal response header.....	200
illegal response status.....	192
method processor.....	203
missing all headers.....	200
missing host header.....	200
missing http protocol.....	205
missing request header.....	201
missing response header.....	201
missing user agent header.....	202
request header overflow.....	202
resource enumeration.....	196
suspicious response status.....	193
unexpected method requested.....	205
unexpected request header.....	203
unexpected response status.....	193

unknown common directory requested.....	194
unknown http protocol.....	206
unknown user directory requested.....	194
user directory enumeration.....	195

Akamai Dynamic Site Accelerator

configure support.....	58
alert service.....	41

appliance

initial configuration.....	31
management.....	21
restart and shutdown.....	109
terminology.....	6
appliance deployment.....	7

applications

edit.....	52
new.....	51
patterns.....	53

assigning the instance and IP

CLI.....	94
verify.....	95
web interface.....	94

Attackers screen.....

autoresponse	
overview.....	143

B

backend server.....	39
requirements.....	39

backend servers

define.....	55
-------------	----

backup

restore.....	117
--------------	-----

backup and recovery

overview.....	115, 117
---------------	----------

backup service.....

fields.....	44
-------------	----

basic configuration mode.....

available sections.....	97
-------------------------	----

braces, in configuration statements.....

brackets

angle, in syntax descriptions.....	xx
square, in configuration statements.....	xxi

C

captcha template.....

CLI

config context.....	48
import/export.....	49
initialize configuration.....	49

overview.....	13, 47	email alerts.....	42
proxy exclusion.....	50	expert configuration mode.....	12
services context.....	50	details.....	98
set config parameter.....	48	F	
system context.....	50	first time configuration.....	11
comments, in configuration statements.....	xxi	font conventions.....	xx
components.....	4	H	
configuration		health check URL.....	116
DNS.....	34	high availability	
first time.....	11	configure.....	65
hostname.....	32	overview.....	17, 65
initialize appliance.....	35	update.....	69
interface.....	33	honeypot processors	
restart network.....	35	access policy processor.....	156
TUI steps.....	31	ajax processor.....	159
web interface.....	11	apache configuration requested.....	163
Configuration		apache password file requested.....	164
Security Monitor.....	139	basic authentication brute force.....	167
configuration wizard.....	12	basic authentication processor.....	162
alert service.....	41	cookie parameter manipulation.....	169
backend server.....	39	cookie processor.....	168
backup service.....	43	hidden input form processor.....	173
confirmation.....	44	hidden link processor.....	175
email alerts.....	42	hidden parameter manipulation.....	174
SMTP servers.....	40	invalid credentials.....	165
Spotlight Secure.....	44	link directory indexing.....	176
conventions		link directory spidering.....	176
text and syntax.....	xx	malicious resource request.....	177
Counter Responses screen.....	135	malicious script execution.....	160
curly braces, in configuration statements.....	xxi	malicious script introspection.....	161
customer support.....	xxi	malicious service call.....	157
contacting JTAC.....	xxi	malicious spider activity.....	179
D		parameter type manipulation.....	173
Dashboard		password cracked.....	166
Security Monitor overview.....	123	protected resource requested.....	166
data sources		query parameter manipulation.....	178
CLI.....	81	query string processor.....	177
DNS settings.....	34	robots processor.....	179
documentation		service directory indexing.....	157
comments on.....	xxi	service directory spider.....	158
E		suspicious file exposed.....	171
EC2 deployment		suspicious filename.....	170
CLI.....	89	suspicious resource enumeration.....	172
overview.....	10	hostname, setting.....	32
web interface.....	90	how it works.....	3
Editor			
overview.....	147		

I

incident methods	
list.....	267
Incidents screen.....	133
initialize appliance.....	35
interface configuration.....	33

L

license	
add.....	37
limitations.....	6
load-balanced environments	
options.....	9
Locations tab.....	132
log format	
counter response.....	276
incident.....	275
overview.....	275
profile.....	277
login ban	
unblock.....	104

M

managing services.....	115
manuals	
comments on.....	xxi
master-slave mode.....	115
multiple web servers	
securing.....	51

N

network placement.....	7
node types.....	15

P

parentheses, in syntax descriptions.....	xxi
password	
change.....	85
reset.....	85
phases of detection and response.....	4
processors	
complexity ratings.....	151
overview.....	25
proxy exclusion.....	50

R

RBAC	
list of groups and roles.....	100

report

CLI.....	79
CLI supported arguments.....	80
data sources.....	81
example.....	82
formatting.....	81
schedule	77
schedule overview.....	27

reporting

overview.....	27
---------------	----

Reports screen.....

139

response processors.....

app vulnerability detected.....	252
application vulnerability processor.....	252
bad captcha answer.....	229
block processor.....	222
captcha answer automation.....	226
captcha cookie manipulation.....	233
captcha directory indexing.....	236
captcha directory probing.....	237
captcha disallowed multipart.....	236
captcha image probing.....	234
captcha parameter manipulation.....	238
captcha request replay attack.....	239
captcha request size limit exceeded.....	235
captcha request tampering.....	231
captcha signature spoofing.....	233
captcha signature tampering.....	232
clippy processor.....	254
csrf parameter tampering.....	245
csrf processor.....	243
csrf remote script inclusion.....	247
expired captcha request.....	230
force logout processor.....	248
Google map processor.....	265
header injection processor.....	248
http referers disabled.....	247
login processor.....	255
mismatched captcha session.....	230
multiple captcha parameter manipulation.....	242
multiple captcha disallow multipart.....	241
multiple captcha replays.....	240
multiple captcha request overflow.....	228
multiple csrf parameter tampering.....	246
no captcha answer provided.....	227
request captcha processor.....	223
site invalid login.....	261
site login brute force.....	264
site login multiple ip.....	262

site login multiple usernames.....	262
site login user brute force.....	264
site login user pooling.....	263
site login user sharing.....	263
site login username scan.....	264
slow connection processor.....	249
strips input processor.....	249
support processor.....	253
unsupported audio captcha requested.....	228
warning code tampering.....	251
warning processor.....	250
Responses tab.....	131
restart networking.....	35
role-based access control.....	99
configure.....	99
S	
Search screen.....	137
secure cluster.....	10, 15, 61
node types.....	15
setup.....	61
update.....	63
security engine incidents.....	152
security processors	
overview.....	153
services.....	5
session cookie tampering.....	152
Sessions screen.....	136
SMTP servers.....	40
Spotlight Secure.....	44
enable.....	119
overview.....	23
SRX integration	
configure.....	73
create filter and terms.....	72
filters and terms overview.....	71
overview.....	19
test.....	107
SSL to client	
enable.....	56
SSL traffic considerations.....	9
statistics.....	112
support, technical See technical support	
syntax conventions.....	xx
system updates.....	109
Systsem Status screen.....	140

T

technical support	
contacting JTAC.....	xxi
third-party load balancer.....	6
tracking processors	
beacon parameter tampering.....	210
beacon session tampering.....	211
client beacon processor.....	209
client classification processor.....	215
client fingerprint processor.....	211
etag beacon processor.....	207
fingerprint directory indexing.....	214
fingerprint directory probing.....	214
fingerprint manipulation.....	215
session etag spoofing.....	208

U

Updates screen.....	140
user preferences.....	103

V

verify connectivity.....	36
verify installation.....	87
verify instance is running.....	95

W

web interface configuration.....	11
whitelist settings.....	57