Junos Pulse Secure Access Service

CIE Best Practices Guide

Release
7.2

Published: 2012-03-20
Part Number: , Revision 1

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

Junos Pulse Secure Access Service CIE Best Practices Guide

Revision History
January 2011—Integrate Version 7.1 new features
March 2012—Integrate Version 7.2 features

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

END USER LICENSE AGREEMENT

Abbreviated Table of Contents

Table of Contents

List of Tables

About This Guide

Document Conventions

Table 1 on page ix 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.

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

Creating CIE-Compatible Web Applications

CHAPTER 1

Creating CIE-Compatible Web Applications

CIE Overview

One of the core technologies that Secure Access offers is the Content Intermediation Engine (CIE), a highly advanced parser and rewriter. The CIE retrieves Web-based content from internal Web servers and changes URL references and Java socket calls so that all network references point to Secure Access.

For instance, when an authenticated user clicks a link, the request goes to Secure Access. Secure Access performs intermediation by parsing the incoming link to determine the internal destination server and then forwarding the request to that internal server on behalf of the end-user. In other words, Secure Access acts as the internal server to the end-user and acts as an end-user to the internal server. This intermediation process provides protection and clear separation between end-users and internal resources.

In order to successfully intermediate Web applications, the CIE must successfully locate all links within a page and rewrite them accurately. This document provides guidelines

to Web application developers and user interface designers for creating Web applications that the CIE can successfully intermediate. The document provides general recommendations, lists the content-types that Secure Access supports, the level of support that Secure Access provides for each of the content types, and the language constructs to avoid.

NOTE: **The Content Intermediation Engine does not intermediate all types of links. For instance, it does not intermediate ftp, rtsp, mms, and mailto links.**

Content Types Supported Through the CIE

The Content Intermediation Engine fully supports Web applications written in standard HTML, JavaScript, VBscript, and Java. There are a few corner cases, however, in which these content types are sensitive to intermediation and parsing. If this document does not contain information about a content type, the Content Intermediation Engine does not officially support it, but the content type may still work through Secure Access.

HTML Support Through the CIE

The Content Intermediation Engine fully supports native HTML 4.0. When creating HTML content, however, please adhere to the guidelines in the following sections.

Use Well-Formed HTML

We recommend that you run your HTML through an HTML syntax checker to ensure that the HTML is well-formed. This process eliminates the possibility of poorly formed HTML with missing information such as end tags and right brackets. Although the Content Intermediation Engine is powerful enough to successfully intermediate invalid HTML, it is safer to write valid and well-structured HTML.

Use Standard HTML

We recommend that you use standard HTML in your Web pages. For example, use the standard format:

```
<A href="www.servername.com:portNo"> Click Here </A>
```

instead of the more rare format:

```
<A href="www.servername.com" port="portNo"> Click Here </A>
```

Specify the Correct Content Type

The **Content-Type** header in your Web page should match the actual content of the document. For example, do not send a content type of **text/html** if the content is XML.

Construct URLs Using RFC Standards

Follow the URL specification available at http://www.faqs.org/rfcs/rfc1738.html when constructing URLs in HTML pages. Avoid using HTML escape codes in the URLs. Use forward slashes ('/') in URLs instead of backward slashes ('\').

Use a Supported HTTP header

The Content Intermediation Engine supports HTTP/1.1 and 1.0 when communicating to the browser, but only supports HTTP/1.0 when communicating to the back-end server. Make sure that all HTTP headers adhere to the HTTP specification for the version that you are employing.

The Content Intermediation Engine does not pass all headers from the internal Web servers to the browser. Avoid using custom headers because the Content Intermediation Engine may not pass those headers back to the browsers. Instead use the standard headers defined by HTTP.

Set Character Encoding Through META Tags

Specify the character set in the **META** tag to avoid problems relating to character encoding. For example, to set the character encoding of a document to EUC-JP, include the following META declaration in the document:

```
<META http-equiv="Content-Type" content="text/html; charset=EUC-JP">
```

Avoid Browser-Specific Code

Avoid writing HTML that is browser-specific. If most commercially-available browsers support a construct, Secure Access probably supports it too. For example, Secure Access supports the following code snippet that uses layers:

```
<style type="text/css">
<!--
#Layer1 {left: 55px; top: 120px;}
#Layer2 {left: 300px; top: 120px;}
-->
</style>
</head>
<body>
<div id="Layer1"><a href="http://www.google.com">Google</a></div>
<div id="Layer2"><a href="http://www.yahoo.com">Yahoo</a></div>
</body>
```

Microsoft Word and Microsoft Power Point can generate Internet Explorer-specific HTML for embedded drawings and figures. The applications embed these tags within comments so that non-Internet Explorer browsers cannot render the tags. The Content Intermediation Engine might not rewrite these conditional comments appropriately.

Do Not Use Multiple BASE Tags

You should only place 1 tags in the HEAD element of your HTML pages—Secure Access ignores any BASE tags that appear inside of the BODY tag of a document. This standard is included in the specifications for HTML 3.2 and later and is enforced by Internet Explorer 7.0 and later. Additionally, you should only include one BASE tag per document, as required by the HTML 3.2 standard and later.

If you have a page that contains multiple BASE tags or BASE tags outside of the HEAD element, you may encounter broken image links or anchors that do not navigate to the proper locations.

Do Not Embed an “<a href” String Within an “<a href” Tag

Do not embed the <a href string within an <a href tag. For example, the following HTML code does not work:

```
document.write(“<a href=”javascript:top.foo(\”<ahref=alink>label link</a>\”)’”);
```

Instead, use variables as shown in this example:

```
document.write(“<script> var str=\” <a href=alink>label link</a>\”;</scr”+”ipt>
<a href=\”javascript:top.foo(str)\”>”);
```

Miscellaneous

In addition to the issues outlined in the previous sections, also keep the following guidelines in mind when creating HTML content:

- Avoid complex nesting and escaping of quotes.
- In HTML tags, do not use null src attributes.
- Avoid using in-line server-side script tags, typically marked by <% ... %>. The server usually processes these tags before they reach the client. Occasionally, when a server does not process the server-side tags, however, the scripts remain on the client page (which can cause problems).
- When writing <OBJECT> and <APPLET> tags, make sure codebase and cabbase, are present.
- For best performance, we recommend that you limit the amount of text between angle brackets < > to less than 10,000 characters. For example:

  ```
  tagName name="To improve performance, break up a very large string
  here."></tagname>.
  ```

- For performance reasons, we recommend that you write pages that contain no more than 4 frames. Exceeding 4 frames can adversely impact Web rewriting performance.

JavaScript Support Through the CIE

The Content Intermediation Engine handles complex uses of JavaScript, including menu animation, field validation, pop-up windows, frame manipulation, and calendar functions. In addition, the Content Intermediation Engine also supports standard and advanced JavaScript functions such as setTimeout, setInterval, and insertAdjacentHTML. When creating JavaScript content, however, please adhere to the guidelines in the following sections.

Use Straightforward JavaScript

Even though the Content Intermediation Engine is sophisticated enough to handle complex constructs in JavaScript, it may have trouble processing code whose purpose is obscured by multiple levels of indirection. We recommend that you write your code in a straightforward fashion in order to enable the Content Intermediation Engine to capture all the URL references.

Usage of document.write

The Content Intermediation Engine supports the use of **document.write**. We recommend the following guidelines when using **document.write**:

- Do not use base href's in **document.write**.
- Avoid writing nested script tags in **document.write**. If you must write nested script tags in a **document.write**, break the string "<script ..." into "<scr" + "ipt...."
- Tags created partially in static HTML and partially in JavaScript are not supported. For example, the following code snippet is not supported. The "textarea" tag is written dynamically using the JavaScript **document.write** function while the rest is written using static HTML

```
<script>
 document.write("<Textarea> Tag contents") ;
</script>
</Textarea>
```

Avoid Complicated Constructs in the eval() Function

The server cannot intermediate JavaScript code that dynamically generates and executes on the browser such as the **eval()** function. Instead, Secure Access inserts a client-side JavaScript parser into the rewritten page in order to parse and rewrite the dynamically generated code. However, the client-side parser is not as sophisticated as the server-side intermediation engine. As a result, Secure Access sometimes accurately rewrites code inside a <script> tag but might not handle the same strings when you pass them through an **eval()** function. Therefore, complicated constructs within an **eval()** function may not work as you expect. For Secure Access, **window.open()** within the **eval()** function works, but accessing the Document Object Model (DOM) in an eval() function might not work.

Do Not Use Common JavaScript Functions on the Left Hand Side of an Assignment Statement

If the javascript code contains an assignment statement where a function call is in the left hand side of an assignment statement then it is not supported through the CIE engine. For Secure Access, foo.setAttribute("bar") = "false" will not work through the CIE engine.

Do Not Assume Element Numbers or Positions in a DOM

The Content Intermediation Engine supports pages that use the Document Object Model (DOM). When traversing the DOM, however, do not assume the number of elements or the position of the elements. Instead, use criteria such as the ID field of the element to access specific DOM elements (since Secure Access may add content to the intermediated page, thereby invalidating the original number of DOM elements). Web pages that assume the number of elements or position of an element may trample upon or use content added by Secure Access.

For example, if you include five elements on a page, Secure Access may add a sixth element to the DOM. When the Web application then attempts to access and display the last element of the page, it displays the element inserted by Secure Access, which was not the desired intent.

This guideline is especially relevant when using the Secure Access toolbar.

Use One Scripting Language Per Page

Use only one scripting language in one page—do not mix JavaScript and VBScript in the same page. If possible, use JavaScript instead of VBScript since VBScript has no IVElished standard that we can recommend at this time.

In relation to scripting languages, keep in mind that using an empty type attribute in a script tag does not work. For example:

```
<script language="javaScript" type=''>
 Some JavaScript Code
</script>
```

Limit usage of the with command

Limit the use of the **with** command. Excessive usage could lead to incorrectly rewritten pages. For example, instead of using:

```
with (doc){
 location=http://...;
}
```

use:

```
doc.location=http://...
```

Even though the Content Intermediation Engine supports the with statement, we recommend that you avoid such statements and use and simpler constructs. Secure Access may not properly rewrite more complicated statements such as nested with statements since it is difficult to distinguish local variable references from property references on an object.

For example:

```
foo = 1;
```

is a local variable but:

```
with (obj) {
 foo = 1;
}
```

In this example, it is difficult to determine if foo is a local variable or a property of obj. Secure Access uses heuristics to trap the common combinations of objects and properties but this practice obviously does not translate to a general solution. For that reason, we recommend that you avoid the use of **with**.

IFRAME Objects Must Contain an src Attribute

IFRAME objects must contain an src attribute to avoid the secure/non-secure warning. For example, the rendering of the following **IFRAME** results in a secure/non-secure warning.

```
var ifrm = document.createElement("IFRAME");
ifrm.id = foo;
ifrm.height = 100;
```

```
ifrm.width = 100;
document.body.insertAdjacentElement("bar",ifrm);
```

Use frames.length Instead of frames[0]

When checking for the existence of frames in a document that may not contain any frames, use **frames.length** instead of **frames[0]**.

Setting a Cookie and Accessing the Cookie Through JavaScript

A cookie is not available through JavaScript unless the HTML body exists in the response to the page where the cookie was set. That is, if you are setting a cookie in an HTML response and want that cookie to be available in JavaScript, the response body must contain some HTML content.

For example, the following web page will not work:

1. Set a cookie, myURL, on a 302 response.
2. The 302 response does not contain any HTML but contains JavaScript.
3. In the **onunload** function in the JavaScript, access the myURL cookie.
4. The cookie is not accessible.

Understand the Number of Cookies You Can Set

Most browsers have an upper bound on the number of cookies that you can set on the client-side through the use of document.cookie. You cannot use the maximum number of cookies allowed by the browser, however, since Secure Access sets cookies as well.

In most deployments, Secure Access manages configuration information by setting up to four cookies. (Depending on the options chosen by the Secure Access administrator, this number might be smaller.) Therefore, your Web application can set the maximum number of cookies allowed by the browser minus four. Deployments that use the eTrust SiteMinder server, however, must set less cookies, since Secure Access sends cookies to the Web browser to enable single sign-on between SiteMinder and Secure Access.

Use Ascii Characters

To render pages through the CIE engine correctly, avoid non-ascii characters such as ` and ñ in JavaScript.

Selective Rewriting Resource Policy for a POST URL

If the **ACTION URL** for a **FORM POST** is being generated on the client-side in JavaScript, a selective rewriting resource policy for the **ACTION URL** may not work.

To work around this issue:

1. Change the web application so that the **ACTION URL** is in static HTML. For example,

   ```
   <FORM method=POST ACTION=http://www.post_server.com>
   ```

2. Change the **POST** to a **GET**.

Comments in Assignment Statements

Comments inserted in the middle of a right hand side assignment statement in javascript are not supported. For example, the following statement is not supported through the CIE engine.

```
foo = foo.replace(/class=*/,'').
//replace(/<p [^>]*>/,'<p>').
replace(/ style=\"\"/,'');
```

Mixing JavaScript and Static Content

Pages where the **OBJECT** tag or the **APPLET** tag is partly written in JavaScript and partly as static content do not function well within the engine. To ensure correct functionality write the complete tag through static text or through JavaScript.

Miscellaneous

In addition to the issues outlined in the previous sections, also keep the following guidelines in mind when creating JavaScript content:

- Avoid using variables that indirectly assign URL references to native JavaScript objects using the array format rather than the regular dot format. For example:

  ```
  document["location"] = "http://www.yahoo.com";
  ```

  and

  ```
  var d = document;
  var l = "location";
  d[l] = "http://www.yahoo.com";
  ```

  Instead, use:

  ```
  document.location = "http://www.yahoo.com";;
  ```

- Do not use HTML and JavaScript reserved words and built-in functions as object names, function names or variable names in your code. For example, do not define and use variables such as **top**, **location**, **pathname**, and **domain**.
- Secure Access occasionally generates its own JavaScript functions that start with the string **Dana**. To avoid conflicts with Secure Access JavaScript functions, avoid using **Dana***XXX* as function and variable names.
- Avoid embedding JavaScript in the src attributes of tags. For example:

  ```
  <frame name="f1" src="JavaScript:func();">
  ```

- Secure Access does not support the use of port in window.location. For example, Secure Access does not support the following JavaScript code:

  ```
  window.location.port = portNo
  ```

Framed Toolbar Support Through the CIE

Secure Access supports two kinds of browsing toolbars, the framed toolbar and the floating toolbar. The framed toolbar displays pertinent information in a frame in the Secure Access end-user console, whereas the floating toolbar floats over the left or right side of the user's browser (possibly obscuring Web content). These toolbars include links to the Secure Access end user home page, a configurable home page, and the end user help system. Additionally, end-users can use the toolbars to sign out of their Secure Access sessions, add bookmarks, and see session expiration information.

If you choose to use the framed toolbar, you must keep certain guidelines in mind when creating your Web application to ensure that the application does not "break" out of the Secure Access frame. The following usage in your Web application could cause the framed toolbar to disappear and display the floating toolbar instead:

- Pop-ups—If your Web application opens up a popup through a window.open call, then the pop-up will not contain a frame. The parent window will continue to display the frame.
- The top variable—We recommend that you do not use the top variable when working with a frame set because after Secure Access intermediates the page, top might reference a different frame than you intend. This change might make the framed toolbar disappear or could cause your intermediated application to work erratically or incorrectly.

  The following example includes a frame set definition that correctly names its frames. The example also shows an example of using target to properly reference a named frame.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
 <HEAD>
  <TITLE>A frameset document</TITLE>
 </HEAD>
 <FRAMESET rows="50%,50%">
  <FRAME name="fixed" src="init_fixed.html">
  <FRAME name="dynamic" src="init_dynamic.html">
 </FRAMESET>
</HTML>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<HTML>
 <HEAD>
  <TITLE>A document with BASE with a specific target</TITLE>
  <BASE href="http://www.mycom.com/Slides" target="dynamic">
 </HEAD>
 <BODY>
  ...beginning of the document...
 </BODY>
</HTML>
```

- The parent variable—You can use the parent variable from within a frame set (see exception that follows), but do not use the parent variable if your Web page does not include a frame set. Also, do not use the parent variable from a JavaScript function within your topmost frame set. If you do, the application does not behave as you intend. Instead, When Secure Access intermediates the page, the variable references the Secure Access frame set instead of your intended document.

CSS Support Through the CIE

The Content Intermediation Engine supports cascading style sheets. When using cascading style sheets, make sure to set their content types to text/css. If you set an incorrect content type, errors could occur through the Content Intermediation Engine. Note that Secure Access does not support JavaScript in cascading style sheets.

Java Support Through the CIE

Java class files contain compiled Java byte-code which the Java Virtual Machine interprets and executes. When Secure Access encounters this byte-code, it rewrites the compiled Java without decompiling it. The Secure Access new byte-code redirects all HTTP(s) and socket based network communication to an intermediate proxy server via secure HTTPS tunneling. This approach provides a secure and portable proxy mechanism for Web-based client/server applications that utilize client Java applets. The Java rewriting technology is available on the Sun JVM (version 1.4.1+) and MS JVM platform

NOTE: **The process of rewriting Java code may affect performance. To improve the performance of Java applications, we recommend using the Enable Java instrumentation caching option in the Maintenance > System > Options page of the Secure Access Web console. For more information, see the *Secure Access Administration Guide*.**

Supported Java Classes and Methods

Secure Access supports most network related classes and methods through the Java rewriting engine. In general, as long as the Java applet uses TCP and the network traffic is initiated from the client, Secure Access supports the applet. The following table lists Java classes and corresponding methods that are supported through the Content Intermediation Engine.

Table 2: Supported Java Classes and Methods

Supported Java class	Correspondingmethods
java.applet.Applet	All methods
java.applet.AppletContext	showDocument
javax.swing.JApplet	All methods

Table 2: Supported Java Classes and Methods *(continued)*

Supported Java class	Correspondingmethods
java.net.Socket	All methods
java.net.URL	getHost, getPort, getFile, getProtocol, openStream, openConnection, toString
java.net.HttpURLConnection	setRequestProperty
java.net.URLConnection	setRequestProperty
java.net.InetAddress	All methods
java.lang.reflect.Method	Invoke
java.lang.Class	getResource
java.lang.ClassLoader	getResource, getResourceAsStream
netscape.javascript.JSObject	eval, call, removeMember, setSlot, setMember
msxml3.IXMLHttpRequest	Open
javax.net.ssl.SSLSocketFactory	createSocket
javax.swing.JEditorPane	setPage
com.ms.lang.RegKey	getStringValue, getIntValue, getBinaryValue
java.util.ResourceBundle	getBundle

Unsupported Java Functionality

Listed below are Java features that are not supported through the Content Intermediation Engine.

- Secure Access may not support class files written in a proprietary format. To prevent Java intermediation problems with Secure Access, ensure that all network-related classes conform to the Sun Java specification. If the class files do not contain standard byte code then Secure Access cannot intermediate the content.
- Secure Access does not support Java applets that include a checksum validation verifying that the applet is unaltered. (Secure Access cannot support this type of validation since it alters the applet's byte code during intermediation.) Instead, you should use the standard code-signing procedures to secure the applet.
- Secure Access does not support Java applets connections that initiate from the server. If the applet contains server-initiated connections through the use of the ServerSocket class, then the applet does not work through Secure Access.

- Secure Access does not support Java applets that make UDP connections.
- Secure Access does not support Java applets that use Java Remote Method Invocation (RMI) Technology.
- Secure Access does not support Java applets that use the Java Web Start architecture (JNLP files).
- Secure Access does not support Java applets that are written for Oracle JVM or IBM JVM.

Code Signing Certificates

Most commercial Java applets that Secure Access intermediates perform privileged tasks. To perform these tasks, the user must accept the certificate that is used to sign the applet. However, since Secure Access modifies the byte-code, the original signature is invalid and Secure Access must re-sign the applet. Secure Access re-signs the applet with a self-signed certificate whose CA is not a trusted root. Due to the use of the self-signed certificate, the browser displays a warning that must be accepted for every launch of the applet. To avoid the frequent security warnings, you need to import a code signing certificate. For instructions, see the *Secure Access Administration Guide*.

Microsoft Silverlight Support

Secure Access Service does not support custom XAP packages hosted on a Sharepoint 2010 that communicate with a back-end server.

However, if you pass URLs as shown below, CIE can rewrite the packages and URLs can be used inside XAP packages:

- Do not generate requests inside assembly files using absolute URLs.
- If URLs are required, pass them using the **initParams** parameter tag.

  When the Silverlight object initializes using the **object** tag from the HTML code, you pass parameters to the object using the **initParams** parameter tag. The following example shows how a Sharepoint 2010 webpage can use this parameter to pass the URL of a video file:

  ```
  <object type=”application/x-silverlight-2” ...>
  <param name="initParams" value="mediaTitle=Media Web
  Part,mediaSource=/Silverlight-Apps/MediaVideoPlayer/Wildlife.wmv,previewImageSource=/Style
   Library/Media Player/VideoPreview.png" />
  ...
  </object>
  ```

  Secure Access Service can create or modify an ActiveX rewriting policy for any URLs similar to the above example.

  Shown below is the rewriting policy for the above parameter:

  ```
  Resource Policies > Web > ActiveX Parameters
  Classid – DFEAF541-F3E1-4C24-ACAC-99C30715084A
  Parameter – initParams:mediaSource,previewImageSource
  Action – Rewrite URL and Response (static HTML only)
  ```

NOTE: The above rewriting policy is included with the Secure Access Service 7.2 software release.

VBScript Support Through the CIE

The Content Intermediation Engine supports VBScript rewriting that complies with the Microsoft VBScript language reference guide. The guide can be found at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/ddfa5183-d458-41bc-a489-070296ced968.asp. VBScript rewriting fails in the following instances:

- If the VBScript is dynamically generated using **document.write**, **document.writeln**, **eval**, or **execScript** then it fails through the CIE engine. In addition, assignment of VBScript code to **innerHTML** and **outerHTML** is not supported through the CIE engine.
- If the VBScript contains code where a function call that the CIE engine rewrites is in the left hand side of an assignment statement then the code does not work through the CIE engine. For example:

  ```
  foo.setAttribute("bar") = "false"
  ```

  does not work through the CIE engine.

Do Not Use Common VBScript Functions on the Left Hand Side of an Assignment Statement

The CIE engine does not support VBScript code where a function call is in the left-hand side of an assignment statement. For example:

```
foo.setAttribute(“bar”) = “false”
```

is not supported. You should rewrite this statement as:

```
foo.setAttribute(“bar”, “false”);
```

ActiveX Support Through the CIE

The Content Intermediation Engine supports Active X programs that do not make network calls (for example, through TCP or HTTP). Active X programs that do make network connections might or might not work through Secure Access. Since a standard is not available that states where URLs, port numbers, or hostnames can be defined, the Content Intermediation Engine may not locate these items and modify them.

For instance, an Active X program could choose to define the first parameter as a URL and the second parameter as the username while another Active X program could reverse the order of parameters. The CIE does not have the necessary knowledge to consistently rewrite the connections in all cases due to the lack of standards inherent to ActiveX.

You can, however, create resource policies that specify parameters that you want to rewrite. These policies must specify the exact URLs and hostnames that the Web page

passes to the Active X controls. For more information, see the *Secure Access Administration Guide*.

Secure Access also supports Active X programs that only contain relative links through the Pass Through Proxy feature.

NOTE: Pages where the OBJECT tag or the APPLET tag is partly written in JavaScript and partly as static content do not function well within the engine. To ensure correct functionality write the complete tag through static text or through JavaScript.

Flash Support Through the CIE

The Content Intermediation Engine supports Flash versions 5, 6, 7 and 8, including dynamic rewriting of internal Web links during an access request. We support the rewriting of Actionscript in Flash. The calls in Actionscript that are supported are: **load**, **play**, **send**, **sendAndLoad**, **loadVariables**, **loadMovie**, **loadVariablesNum**, **loadMovieNum**, **loadClip**, **loadSound**, **apply**, **connect** on classes of **XML**, **Sound**, **MovieClip**, **NetConnection**, and **MovieClipLoader**. The **eval** equivalent of Actionscript is not supported. Therefore we recommend that the above function calls not be embedded in an Actionscript string object. Note, however, that Secure Access does not support Flash applications that use the **XMLSocket** object or Flash remoting.

If an assignment statement is used for URLs in objects then it will not work through the rewriter.

For example, the following construct is not supported:

```
xml_connector_obj.URL = "http://....";
xml_connector_obj.trigger();
```

XML Support Through the CIE

Secure Access supports Web applications that use DTDs, XML schemas, and XML islands within an HTML file. When creating XML content, however, please note the following guidelines:

- Secure Access does not support referencing style sheets or DTDs on a separate server.
- Secure Access does not support using the document call.
- Secure Access does not support using the CSS extension for the Microsoft alpha image loader when rewriting XSLT style sheets. However, you can use the alpha image loader if you do not invoke XSLT expressions. For example, the CIE does not support the STYLE portion of the following code:

```
<!-- This DIV is the target container for the filter. -->
<DIV ID="oDiv" STYLE="position:relative; width:200px; color:gold;
filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(
src='/workshop/graphics/earglobe.gif');" >
The World
```

```
</DIV>
```

- Secure Access does not support XSL style sheets that use Microsoft WD-XSL and that use XSL expressions to construct hyperlinks. Secure Access delivers your page correctly, however, provided all hyperlinks within the page do not use XSL expressions.
- Secure Access does not support passing the parameters of ActiveX or Applet objects using XSLT expressions. If you do not use HTML hyperlinks, however, the objects function properly.
- Secure Access does not support manipulating DTD, Xlink, XForm, and XInclude using XSLT expressions. If you do not use XSLT expressions when creating these, however, the page functions properly through Secure Access.
- Secure Access does not support rewriting DTDs inside an HTML file.
- Secure Access does not support using XSLT expressions to generate HTML hyperlinks inside JavaScript or VBScript statements. As long as XSLT expression is not used to generate a hyper link inside javascript or VBScript, the page functions properly.

PDF Support Through the CIE

The Content Intermediation Engine supports rewriting PDF files from all Acrobat versions when you enable the Rewrite links in PDF files option on the Users > User Roles > *Role* > Web > Options page of the Secure Access Web console. When you select this option, Secure Access rewrites absolute URLs (such as http://www.google.com) and relative URLs (such as http://yourcompany.intranet.net/images/../test.gif). Otherwise, if you do not select this option, Secure Access may not properly display PDF files with links.

Secure Access supports rewriting normal PDFs and linearized PDFs. A *normal* PDF requires the browser download the entire document before displaying it. A *linearized* PDF enables the browser to download parts of the document separately, thereby allowing the browser to start displaying the document before it is completely downloaded.

NOTE:

- Secure Access does not rewrite embedded streams in PDF files.
- Secure Access does not modify encrypted or digitally signed PDF files at all.
- Manually edited PDF files that have incorrect byte offsets do not work correctly through Secure Access. Even though these files might work through Acrobat 7, they are not supported through Secure Access.
- PDF files that contain 2 objects for the same link do not work through Secure Access. You can check if the PDF file contains two objects for the same link by doing the following:

  1. Open the PDF with Acrobat with Notepad or Wordpad and look for the URI for which you would like to determine the object. (Open the PDF file with Notepad or Wordpad instead of the Acrobat Reader.)
  2. Look for the URI string and find out what is the object number that references this URI. The URI object is in the following format:

     ```
     55 0 obj
     <</S URL
     ...
     /URI (http://www.google.com)>>
     endobj
     ```

     where 55 is the object number.
  3. Next check if the file contains another object with the same object number referencing a different URL. If another object with the same number is found then the PDF file cannot rewritten through the CIE engine.

Streaming Media and Video Content

Since streaming media content often contains direct network connections without the use of HTTP, the CIE cannot support it. If you deliver the streaming content through an <OBJECT> tag and one of the attributes of the tag is a URL to which an HTTP connection is made, then the content may work through Secure Access.

Content Types Supported Through Pass Through Proxy

Pass Through Proxy is a key component of the Content Intermediation Engine that supports various intermediation-sensitive content types with relative links such as Active X, the IBM JVM, and the Oracle JVM.

Determining When to Use the CIE vs. Pass Through Proxy

Our final recommendation is to test the Web-based content through the Secure Access Content Intermediation Engine. If a page does not display accurately then this document

can provide suggestions on how you can alter your code to ensure compatibility with Secure Access.

To summarize, Web applications written in HTML, JavaScript, VBScript, Java, or XML that use the guidelines listed in this document should work seamlessly with the Secure Access Content Intermediation Engine (CIE). Secure Access supports other content types such as Active X through the Pass Through Proxy feature as long as the content only contains relative links.

Table 3: Supported Content Types

Content Type	Support Level
HTML	Full Supported
JavaScript	Full Supported
VBScript	Full Supported
Java	Full Supported
Flash	Full Supported
ActiveX	Partially Support*
PDF	Partially Supported
XML	Full Supported
Streaming	Very limited support. If your application does not work, please contact your account team about using Network Connect as an alternative.

* To use partially supported formats, you may need to use the Pass Through Proxy option. (Refer to this document for details.) In most cases, Secure Access can intermediate the content, and in the few cases where it cannot, you can easily modify the content to a supportable format. If you cannot modify the content, please contact your account team about using the Secure Application Manager as an alternative

PART 2

Index

Index