Juniper Networks vSRX

RECOMMENDED

OVERVIEW

In Q4 2023, CyberRatings.org conducted an independent test of Juniper Networks vSRX Next-Generation Virtual Firewall (vSRX Junos 22.4R2.8) against the Cloud Firewall Test Methodology v2.1, using Amazon Web Services running a c5n.2xlarge instance. The product was thoroughly tested to determine how it handled TLS/SSL 1.2 and 1.3 cipher suites, how it defended against 984 exploits, whether any of 1,645 evasions could bypass protection, and if the device would remain stable under adverse conditions. To provide a more realistic rating based on modern network traffic, both clear text and encrypted traffic were measured.

99.70% SECURITY EFFECTIVENESS

Routing & Access Control	100.00%
TLS/SSL Functionality	100.00%
Threat Prevention	99.70%
Stability & Reliability	100.00%

ROUTING & ACCESS CONTROL

Unrestricted Traffic Test	Pass
Segmented Traffic Test	Pass
Simple Policies	Pass
Complex Multi-Zone Policies	Pass

TLS/SSL FUNCTIONALITY

Decryption Validation	Supported
Top 10 Cipher Support	10/10 Supported
Prevention of Weak Ciphers	5/5 Prevented
Decryption Bypass Exceptions	Supported
TLS Session Reuse - Session Tickets	Supported
TLS Session Reuse - Session IDs	Supported

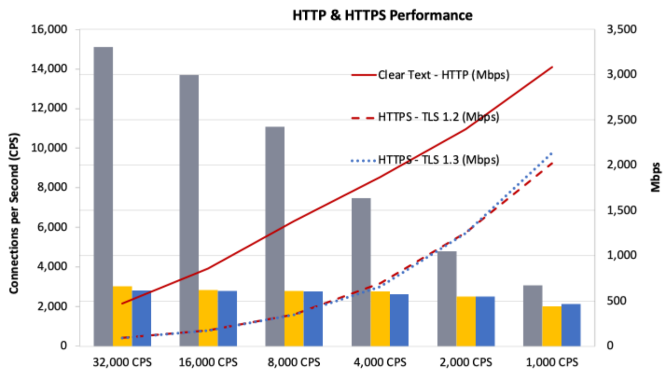
STABILITY & RELIABILITY

Protocol Fuzzing & Mutation	Pass
Blocking with Minimal Load	Pass
Blocking Under Load	Pass
Attack Detection/Blocking – Normal Load	Pass
State Preservation – Normal Load	Pass
Pass Legitimate Traffic – Normal Load	Pass
State Preservation – Maximum Exceeded	Pass
Drop Traffic – Maximum Exceeded	Pass

THREAT PREVENTION

False Positives	Pass (233/233)
Exploits	981/984
Evasions	1,645/1,645

RATED THROUGHPUT – 1,228 MBPS



Clear Text (HTTP)		TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30) TLS 1.2		TLS_AES_256_GCM_SHA384 (0x13, 0x02) TLS 1.3	
CPS	Mbps	CPS	Mbps	CPS	Mbps
15,100	472	3,020	94	2,827	88
13,700	856	2,850	178	2,803	175
11,085	1,386	2,802	350	2,774	347
7,468	1,867	2,778	695	2,632	658
4,800	2,400	2,515	1,258	2,502	1,251
3,085	3,085	2,020	2,020	2,137	2,137

MSRP + 24/7 SUPPORT & MAINTENANCE

Juniper Networks 1-Year Cost	$3,224.00
Amazon Web Services (AWS)	$3,784.32

Routing & Policy Enforcement

Access control is the primary responsibility of a firewall. Firewalls have undergone several stages of development, from early packet filtering and circuit relay firewalls to application-layer (proxy-based), dynamic packet filtering firewalls, and user/application-aware "next-generation" firewalls. Throughout its history, the goal has been to enforce an access control policy between two networks. Rules were configured to permit or deny traffic from one network resource to another based on identifying criteria such as source IP, destination IP, source port, destination port, and protocols.

This test validates that the firewall enforces security policies over a range of policy environments, from simple to complex. The tests incrementally build on a baseline consisting of a simple configuration with no policy restrictions and no content inspection – to a complex multiple-zone configuration that supports many users, networks, policies, and applications. At each level of complexity, traffic was tested to ensure specified policies were enforced.

Network Segmentation	Results
Unrestricted Traffic Test (Allow All)	Pass
Segmented Traffic Test	Pass
Access Control	Results
Simple Policies	Pass
Complex Multi-Zone Policies	Pass

Figure 1 – Routing & Policy Enforcement

TLS/SSL Functionality

The use of the Secure Sockets Layer (SSL) protocol and its current iteration, Transport Layer Security (TLS), is now the norm. Let's Encrypt statistics show that as of December 2023, over 80% of web traffic was sent over HTTPS.[1]

While CyberRatings believes using encryption is good, TLS/SSL is susceptible to various security attacks at multiple levels of network communication. For example, attacks have been observed in the handshake protocol, record protocol, application data protocol, and Public Key Infrastructure (PKI). To address the growing threat of focused attacks using the most common web protocols and applications, the capabilities of the DUT was tested to provide visibility into the TLS/SSL payloads and detect attacks concealed by encryption and attacks against the encryption protocols themselves. The table below lists the tested TLS/SSL in order of prevalence[2] per December 2023.

Decryption Validation

Version	Prevalence	Cipher Suites	Results
TLS 1.3	66.51%	TLS_AES_256_GCM_SHA384 (0x13, 0x02)	Pass
TLS 1.2	11.85%	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)	Pass
TLS 1.2	9.26%	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)	Pass
TLS 1.3	8.07%	TLS_AES_128_GCM_SHA256 (0x13, 0x01)	Pass
TLS 1.2	1.72%	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA8)	Pass
TLS 1.2	0.68%	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28)	Pass
TLS 1.3	0.55%	TLS_CHACHA20_POLY1305_SHA256 (0x13, 0x03)	Pass
TLS 1.2	0.42%	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C)	Pass
TLS 1.2	0.27%	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA9)	Pass
TLS 1.2	0.20%	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B)	Pass

Figure 2 – TLS/SSL Functionality

First, we tested how the firewall handled cipher suites known to be insecure, using null ciphers (no encryption of data) and anonymous ciphers (no authorization). Then we validated the ability to correctly decrypt and inspect TLS/SSL traffic using prohibited content previously blocked during testing. The content was then encrypted and verified that it was still blocked. We then tested to see if we could permit conditional bypass of decryption. This might be required to preserve privacy for regulatory or other reasons.

Decryption Validation	Supported
Top 10 Cipher Support	10/10 Supported
Prevention of Weak Ciphers	5/5 Prevented
Decryption Bypass Exceptions	Supported
TLS Session Reuse - Session Tickets	Supported
TLS Session Reuse - Session IDs	Supported

[1] Let's Encrypt Stats (https://letsencrypt.org/stats/)
[2] https://crawler.ninja/files/ciphers.txt

Threat Prevention

A firewall is a mechanism used to protect a trusted network from an untrusted network while allowing authorized communications to pass from one side to the other, thus facilitating secure business use of the Internet. The CyberRatings exploit repository contains exploits demonstrating many protocols and applications. Exploits are selected based on CVSS score (how widely used is an application + what can an attacker do?), use case, and customer relevance.

False Positives

A key to effective protection is correctly identifying and allowing legitimate traffic while maintaining protection. False positives are any legitimate, non-malicious content/traffic perceived as malicious. False positive tests verify the ability of the firewall to block attacks while permitting legitimate traffic. If a device experienced false positive events, it was tuned until no further false positive events were encountered.

Exploit Protection

99.70% Blocked (981/984)

An exploit is an attack that takes advantage of a protocol, product, operating system, or application vulnerability. CyberRatings verified that the firewall could detect and block exploits while remaining resistant to false positives by attempting to send exploits through the product under test and verified that the malicious traffic was blocked, and all appropriate logging and notifications were performed.

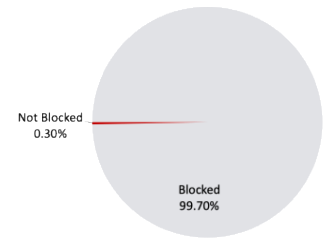


Figure 3 - Exploit Effectiveness

Coverage by Date

Figure 4 provides insight into whether a vendor is aggressively aging out protection signatures to preserve performance levels. It also reveals whether a product lags in protection for the most current vulnerabilities. CyberRatings reports exploits by individual years for the past ten years.

Year	Coverage %
<=2014	100%
2015	100%
2016	100%
2017	100%
2018	100%
2019	99.46%
2020	100%
2021	100%
2022	100%
2023	98.46%

Figure 4 - Coverage by Date

Coverage by Target Vendor

Exploits within the CyberRatings exploit library target many protocols and applications. The below figure shows how the product under test offers exploit protection for ten top vendors targeted in this test.

Vendor	Coverage %
Adobe	100%
Advantech	100%
Apache	100%
IBM	100%
Cisco	100%
HPE	100%
Microsoft	100%
Oracle	100%
SolarWinds	100%
VMware	100%

Figure 5 – Coverage for Top Vendors

Resistance to Evasions 100% Effective (1,645/1,645)

Threat actors apply evasion techniques to disguise and modify attacks to avoid detection by security products. An attacker can bypass protection if a firewall fails to detect a single form of evasion. Therefore, it is imperative that a firewall correctly handles evasions.

Our engineers verified that the firewall could block exploits when subjected to numerous evasion techniques. To develop a baseline, we took several previously blocked attacks. We then applied evasion techniques to those baseline samples and tested them. This ensured that any misses were due to the evasions, not the baseline samples.

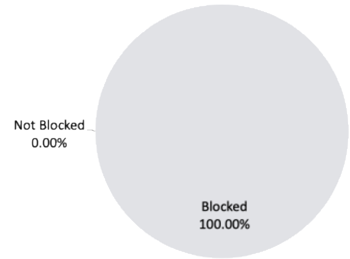


Figure 6 – Evasion Effectiveness

We adjusted scoring for evasions according to their impact: For example, TCP evasions are more impactful than HTML evasions. A TCP evasion can be applied to thousands of exploits, vs. an HTTP evasion that is limited to far fewer exploits.

During testing, we used multiple exploits for each evasion technique to see how each product defended against these combinations. Some products properly handled an evasion technique with all tested exploits while others handled evasions with only some of the exploits.

Evasion Technique	Evasions Tested	Evasions Blocked
IP Packet Fragmentation	162	162
JSON Obfuscation	2	2
TCP Transfer Control Block (TCP Split-Handshake)	9	9
IP Address Spoofing	18	18
TCP Stream Segmentation	468	468
HTTP Chunked Encoding	171	171
HTTP Compression	36	36
HTTP Headers	117	117
Layered Evasions		
HTTP Compression, HTTP Chunked Encoding	207	207
HTTP Chunked Encoding, HTTP Headers	72	72
HTTP Compression, HTTP Chunked Encoding, TCP Stream Segmentation	36	36
HTTP Compression, HTTP Chunked Encoding, TCP Stream Segmentation, IP Packet Fragmentation	36	36
IP Packet Fragmentation, IP Insertion	99	99
JSON Obfuscation, HTTP Compression, HTTP Chunked Encoding, TCP Stream Segmentation	16	16
JSON Obfuscation, HTTP Compression, HTTP Chunked Encoding, TCP Stream Segmentation, IP Packet Fragmentation	16	16
TCP Stream Segmentation, IP Packet Fragmentation	36	36
TCP Stream Segmentation, TCP Insertion	72	72
TCP Stream Segmentation, TCP Transfer Control Block	72	72

Figure 7 – Evasions by Technique

Performance

The performance of the cloud network firewall was tested using various traffic conditions that provide metrics for real-world performance. Individual implementations will vary based on usage; however, these quantitative metrics provide a gauge as to whether a particular firewall is appropriate for a given environment.

Amazon Web Services (AWS) offers a wide range of instance types with baseline and burst bandwidth[3]. Depending on the instance type used by the vendor, the same instance was used to measure baseline control, and then repeated with the firewall in the exact same configuration for exploits and evasions.

Rated Throughput

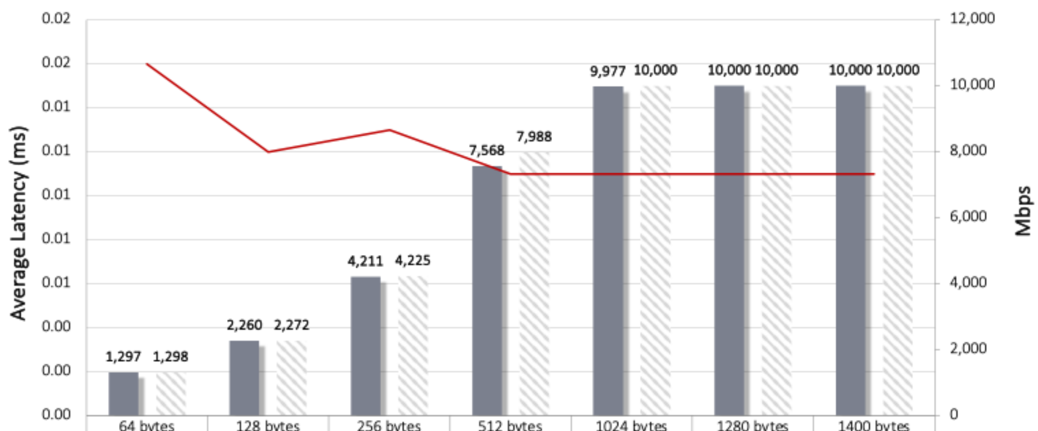
We measured performance to determine the sustained throughput of the cloud network firewall over time for a range of packet sizes and connections per second to capture the firewall's performance curves for UDP, HTTP, and HTTPS. The "Plain Text Rated Throughput," "HTTPS Rated Throughput," and the combined "Rated Throughput" are good benchmarks for what an enterprise can expect the firewall instance to achieve consistently (over time) when deployed on AWS.

Performance		Result
Plain Text Rated Throughput (Average of HTTP capacity — without delays)	1,678 Mbps	Rated Throughput **1,228 Mbps**
HTTPS (TLS/SSL) Rated Throughput (Average of HTTPS Capacity tests)	779 Mbps	

Figure 8 – Rated Throughput (Mbps)

Raw Packet Processing Performance (UDP Throughput)

UDP packets of varying sizes were generated with variable source and destination IP addresses transmitting from a fixed source port to a fixed destination port. This allowed us to create a constant stream for each packet size tested (e.g. 64 bytes) transmitted bidirectionally through the firewall. This was to determine the maximum rate the firewall could process packets of various sizes, and the associated latency. Each vendor was required to write a signature to detect the test packets to ensure that they were being passed through the detection engine and not "fast-tracked" from the inbound port to the outbound port.



	64 bytes	128 bytes	256 bytes	512 bytes	1024 bytes	1280 bytes	1400 bytes
Juniper Tested Throughput (Mbps)	1,297	2,260	4,211	7,568	9,977	10,000	10,000
AWS Maximum Throughput (Mbps)	1,298	2,272	4,225	7,988	10,000	10,000	10,000
Juniper Average Latency (ms)	0.02	0.01	0.01	0.01	0.01	0.01	0.01

Figure 9 - Raw Packet Processing Performance (UDP Traffic)

[3] https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/compute-optimized-instances.html and https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/memory-optimized-instances.html

Theoretical Maximum Capacity

The goal was to stress the firewall and determine how it handles high volumes of TCP connections per second, HTTP transactions per second, and concurrent open connections. All packets contained valid payload and address data, and these tests provided an excellent measurement of maximum connection rates and concurrency (simultaneous users/traffic).

- **Theoretical Maximum Concurrent TCP Connections:** This type of traffic would not typically be found on a network, but it provides the means to determine the maximum possible concurrent connections.
- **Maximum HTTP Transactions per Second**: This test is designed to determine the maximum HTTP transaction rate of the firewall with a one-byte response size. The object size defines the number of bytes in the body, excluding any bytes associated with the HTTP header. A one-byte response size is designed to provide the theoretical maximum HTTP transactions per second rate.

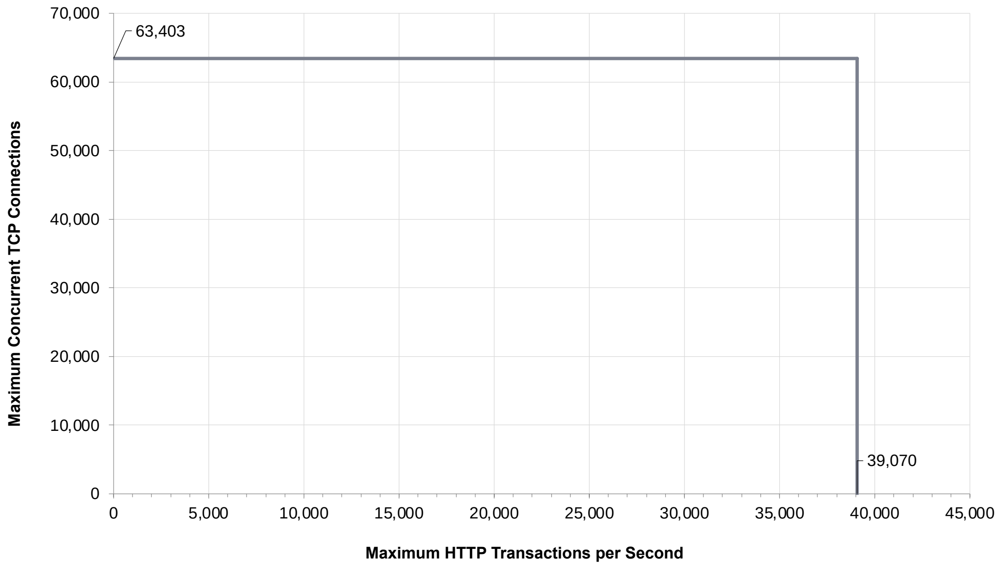
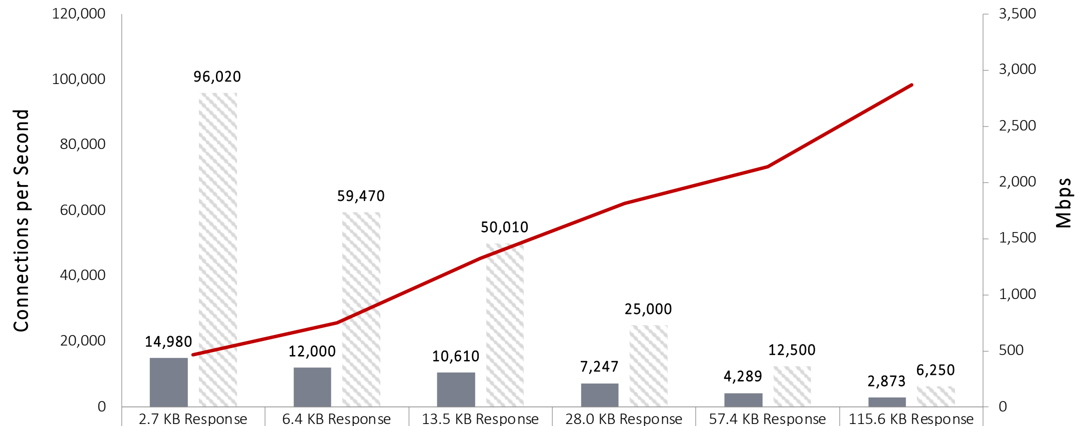


Figure 10 – Max TCP Concurrency and HTTP Transactions per Second

HTTP Capacity

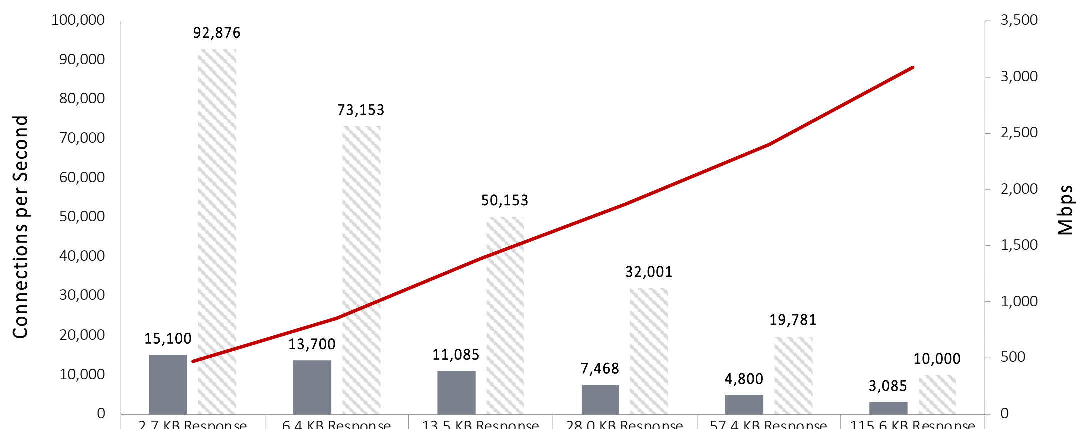
The goal was to stress the HTTP detection engine and determine how the device copes with network loads of varying average packet sizes and varying connections per second. By creating genuine session-based traffic with varying session lengths, the device was forced to track valid TCP sessions, thus ensuring a higher workload than simple packet-based background traffic. This provided a test environment as close to real-world conditions as possible in a lab while ensuring absolute accuracy and repeatability.



	2.7 KB Response	6.4 KB Response	13.5 KB Response	28.0 KB Response	57.4 KB Response	115.6 KB Response
Juniper Tested Throughput (CPS)	14,980	12,000	10,610	7,247	4,289	2,873
AWS Maximum Throughput (CPS)	96,020	59,470	50,010	25,000	12,500	6,250
Juniper Tested Throughput (Mbps)	468	750	1,326	1,812	2,145	2,873

Figure 11 - HTTP Capacity (with delays)

Each transaction consisted of a single HTTP GET request with delays (i.e., the web browser/client waits ten seconds to "read" the content provided by the web server. The web server then responds immediately, after the web browser/client clicks to the next page thus maintaining each connection for ten seconds). All packets contained valid payloads.



	2.7 KB Response	6.4 KB Response	13.5 KB Response	28.0 KB Response	57.4 KB Response	115.6 KB Response
Juniper Tested Throughput (CPS)	15,100	13,700	11,085	7,468	4,800	3,085
AWS Maximum Throughput (CPS)	92,876	73,153	50,153	32,001	19,781	10,000
Juniper Tested Throughput (Mbps)	472	856	1,386	1,867	2,400	3,085

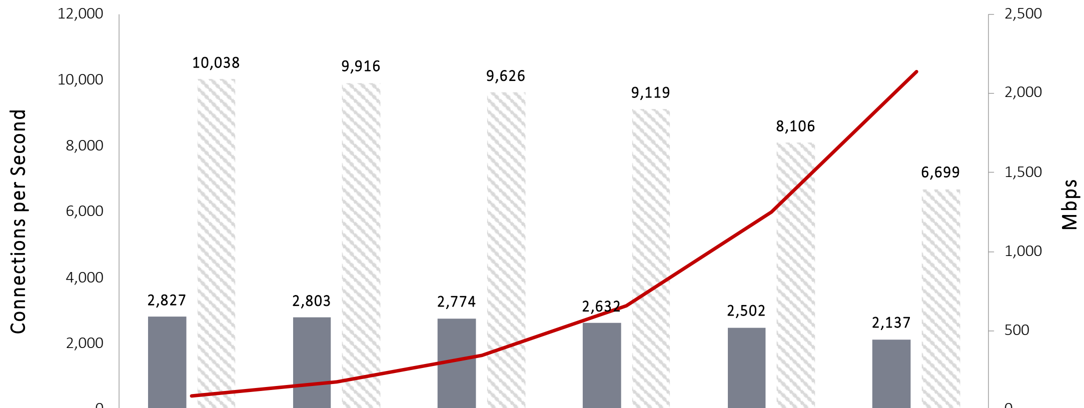
Figure 12 - HTTP Capacity (without delays)

Each transaction consisted of a single HTTP GET request, and there were no delays (i.e., the webserver responded immediately to all requests). All packets contained valid payload (a mix of binary and ASCII objects) and address data. Testing determined the maximum rate the firewall was able to process HTTP packets of multiple sizes and its efficiency at forwarding packets quickly to provide the highest level of network performance with the lowest latency. The results were recorded at each response size at a load level of 95% of the maximum throughput, just before latency increased (which indicates the throughput is not sustainable).

TLS/SSL Capacity

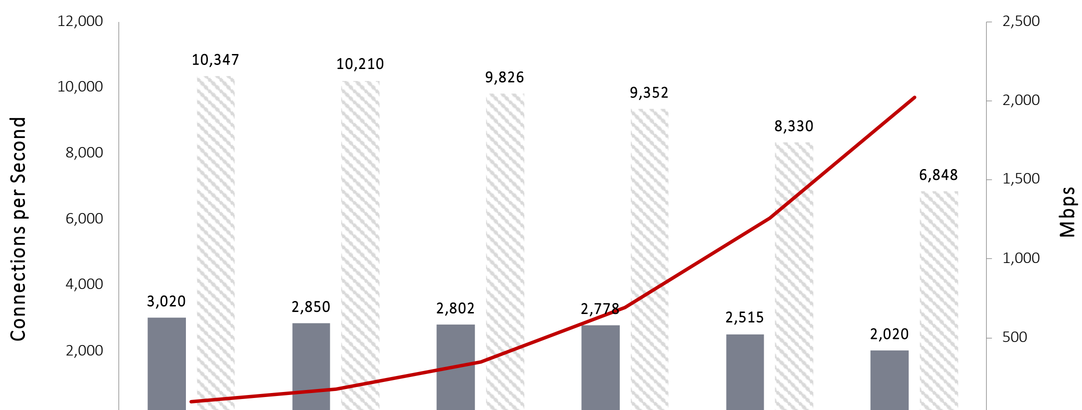
The goal was to stress the HTTPS engine and determine how the device coped with network loads of varying average packet sizes and varying connections per second.

By creating session-based traffic with varying session lengths, the device was forced to track valid TCP sessions, thus ensuring a higher workload than simple packet-based background traffic. Encrypting the traffic using TLS/SSL with varying algorithms forced the device to decrypt traffic before inspection, increasing the workload further. This provided a test environment that is as close to real-world conditions as possible to achieve in a lab environment (albeit biased towards HTTPS traffic) while ensuring accuracy and repeatability. Tests were performed similarly to HTTP with one HTTPS transaction per connection. Testing determined the maximum rate the firewall was able to process HTTPS traffic of various sizes and its efficiency at forwarding packets quickly to provide the highest level of network performance with the lowest latency. The results were recorded at each response size at a load level of 95% of the maximum throughput, just before latency increased (which indicates the throughput is not sustainable).



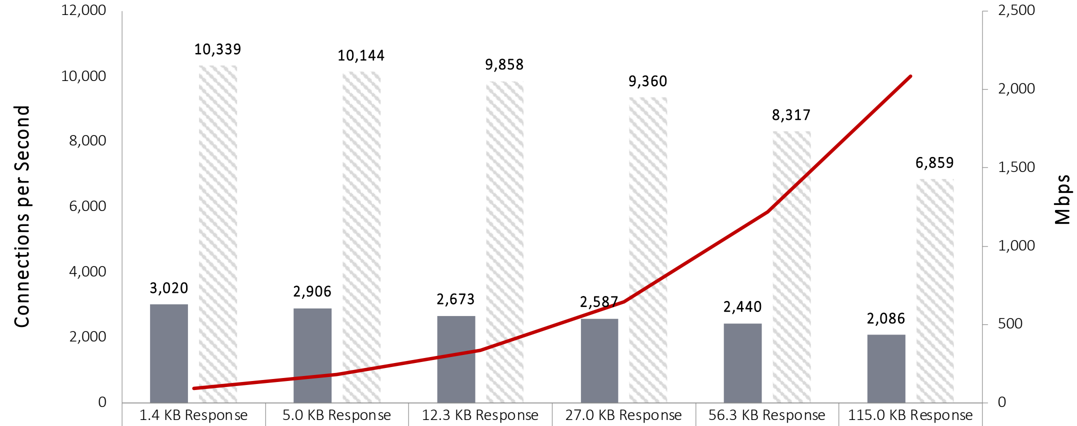
	0.2 KB Response	3.9 KB Response	11.2 KB Response	25.7 KB Response	54.9 KB Response	113.8 KB Response
Juniper Tested Throughput (CPS)	2,827	2,803	2,774	2,632	2,502	2,137
AWS Maximum Throughput (CPS)	10,038	9,916	9,626	9,119	8,106	6,699
Juniper Tested Throughput (Mbps)	88	175	347	658	1,251	2,137

Figure 13 - HTTPS Capacity for TLS 1.3 (TLS_AES_256_GCM_SHA384 [0x13, 0x02])



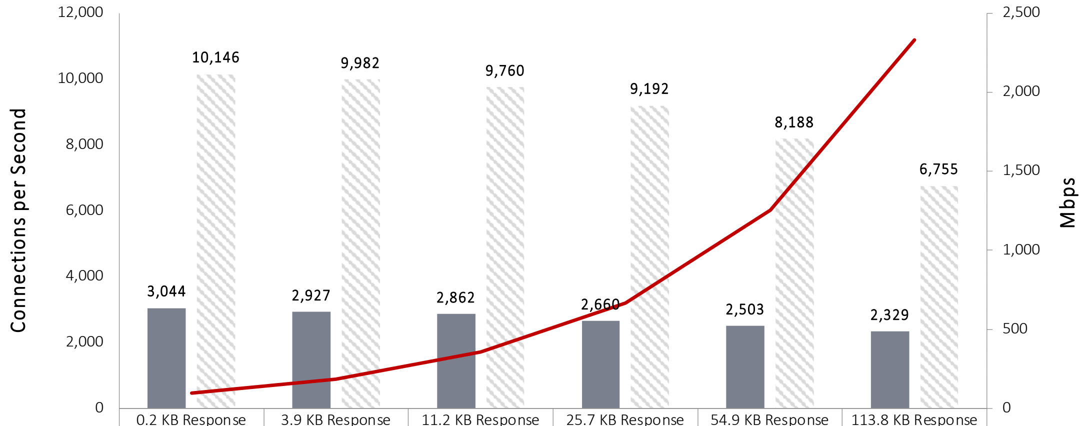
	1.4 KB Response	5.0 KB Response	12.3 KB Response	27.0 KB Response	56.3 KB Response	115.0 KB Response
Juniper Tested Throughput (CPS)	3,020	2,850	2,802	2,778	2,515	2,020
AWS Maximum Throughput (CPS)	10,347	10,210	9,826	9,352	8,330	6,848
Juniper Tested Throughput (Mbps)	94	178	350	695	1,258	2,020

Figure 14 - HTTPS Capacity for TLS 1.2 (TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 [0xC0, 0x30])



	1.4 KB Response	5.0 KB Response	12.3 KB Response	27.0 KB Response	56.3 KB Response	115.0 KB Response
Juniper Tested Throughput (CPS)	3,020	2,906	2,673	2,587	2,440	2,086
AWS Maximum Throughput (CPS)	10,339	10,144	9,858	9,360	8,317	6,859
Juniper Tested Throughput (Mbps)	94	182	334	647	1,220	2,086

Figure 15 - HTTPS Capacity for TLS 1.2 (TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 [0xC0, 0x2F])



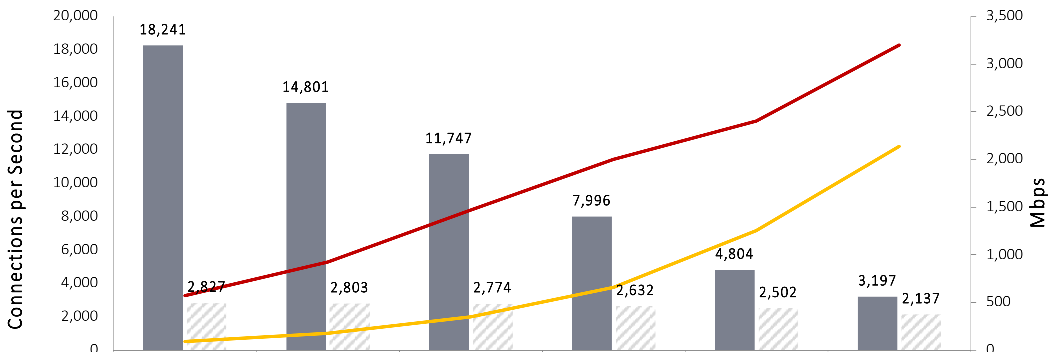
	0.2 KB Response	3.9 KB Response	11.2 KB Response	25.7 KB Response	54.9 KB Response	113.8 KB Response
Juniper Tested Throughput (CPS)	3,044	2,927	2,862	2,660	2,503	2,329
AWS Maximum Throughput (CPS)	10,146	9,982	9,760	9,192	8,188	6,755
Juniper Tested Throughput (Mbps)	95	183	358	665	1,252	2,329

Figure 16 - HTTPS Capacity for TLS 1.3 (TLS_AES_128_GCM_SHA256 [0x13, 0x01])

Delta between HTTP and HTTPS Capacity & Throughput

How did the encryption overhead affect the bandwidth for the provided payloads? And how does the size of what is being transferred impact performance?

The purpose of these tests was to measure the amount of overhead added to each payload based on the cipher suite used. This test used HTTP without any TLS and then we tested the same payload using TLS 1.3 (TLS_AES_256_GCM_SHA384 [0x13, 0x02]). Each transaction consisted of a single HTTPS GET request with no transaction delays (i.e., the web server responds immediately to all requests). All traffic contains valid payloads.



	0.2 KB	3.9 KB	11.2 KB	25.7 KB	54.9 KB	113.8 KB
Juniper HTTP Throughput (CPS)	18,241	14,801	11,747	7,996	4,804	3,197
Juniper HTTPS Throughput (CPS)	2,827	2,803	2,774	2,632	2,502	2,137
Juniper HTTP Throughput (Mbps)	570	925	1,468	1,999	2,402	3,197
Juniper HTTPS Throughput (Mbps)	88	175	347	658	1,251	2,137

Figure 17 – Delta between HTTP and HTTPS Capacity & Throughput

Stability and Reliability

Long-term stability is essential for a firewall, where failure can produce network outages. These tests verified the firewall's stability and ability to maintain security effectiveness while under normal load and while passing malicious traffic. Products that could not sustain legitimate traffic (or that crash) while under hostile attack did not pass.

The product was required to remain operational and stable throughout these tests and to block 100% of previously blocked traffic, raising an alert for each. If any policy-forbidden traffic passed, caused by either the volume of traffic or by the product failing open for any reason, this resulted in a fail.

Stability and Reliability	Result
Protocol Fuzzing & Mutation	Pass
Blocking under Extended Attack	
Blocking with Minimal Load	Pass
Blocking Under Load	Pass
Behavior of the State Engine under Load	
Attack Detection/Blocking – Normal Load	Pass
State Preservation – Normal Load	Pass
Pass Legitimate Traffic – Normal Load	Pass
State Preservation – Maximum Exceeded	Pass
Drop Traffic – Maximum Exceeded	Pass

Protocol Fuzzing & Mutation

Testing determined how the firewall responded (e.g., crashes, reboots, etc.) due to traffic generated from various protocol randomizers and mutation tools. The product was expected to remain operational and capable of detecting and blocking exploits throughout the test.

Blocking Under Extended Attack

These tests indicated the ability of the firewall to remain operational and stable (i.e., block violations and raise associated alerts) throughout an extended attack.

Behavior of the State Engine Under Load

These tests determined whether the device could preserve its state across a large number of open connections over an extended period. At various points throughout the test (including after the maximum had been reached), it was confirmed that the device could inspect and block traffic that violated the currently applied security policy while ensuring that legitimate traffic was not blocked.

Total Cost of Ownership

When calculating TCO for a cloud firewall, there are several considerations:

- **First**, there is the cost of the cloud provider and the specific price for the cloud firewall instance.
- **Second**, some instances offer a guaranteed level of throughput; others offer boosts up to a certain amount of throughput but often fail to specify what type of traffic for a given period.
- **Third**, there is the ongoing cost of running the instance (cost per hour), which can be different for each region selected.

Traditional licenses are offered, but so are bundles, which could be as long as three or five years. There are also pay-as-you-go options, which are charged hourly or daily. Furthermore, enterprises should include labor costs for operational expenditures (OPEX) such as administration, policy and configuration handling, log handling, alert handling, monitoring, reporting, analysis, auditing and compliance, maintenance, software updates, and troubleshooting.

In order to reduce complexity and provide meaningful guidance, we held the labor and OPEX costs constant. This yielded a simplified formula to measure the TCO and value of the firewall:

Security Effectiveness = Exploit Block Rate* Evasions* Stability and Reliability[4]

TCO per Protected Mbps = TCO / (Security Effectiveness * Tested Throughput)

Figure 18 – Security Effectiveness and TCO per Protected Mbps Formulas

This formula incorporates the cost of the cloud firewall, the instance costs, and how effective the firewall is in delivering both security and performance over time. The *TCO per Protected Mbps* metric provides clear guidance on whether a product's price is higher or lower than its competitors.

Figure 19 contains the pay-as-you-go pricing for Juniper Networks Next-Generation Firewall. It is not meant as an extensive list. The pricing was collected from Amazon Web Services (AWS)[5].

vCPU/Instance	Juniper Networks License	License Cost 1 year	License Cost 3 years	License Cost 5 years
c5n.2xlarge	vSRX Next Generation Virtual Firewall	$3,224.00	$9,672.00	$16,120.00

Figure 19 – Juniper Networks List Price

If a customer opts to use Juniper Networks annual pay-as-you-go license and the c5n.2xlarge in AWS (North-Virginia), with an hourly cost of $0.432, then the calculation would be as follows:

	Cost c5n.2xlarge	Daily Cost	Annual Cost	AWS Cloud + Juniper Networks Cost
AWS	$0.432	$10.37	$3,784.32	$7,008.32
Juniper Networks			$3,224.00	

Now, we can calculate the TCO:

AWS Cloud + Juniper Networks	Exploit Block Rate	Evasions	Stability & Reliability	Tested Throughput	TCO per Protected Mbps
$7,008.32	99.70%	100%	100%	1,228	$5.72

Figure 20 – TCO Calculation

[4] Stability and Reliability in this formula includes TLS/SSL functionality.
[5] https://aws.amazon.com/marketplace/pp/prodview-z7jcugjx442hw?sr=0-1&ref_=beagle&applicationId=AWSMPContessa

Appendix A – Scorecard

Summary			
Vendor		Juniper	
Cloud Service Provider		AWS	
AWS Instance Type		c5n.2xlarge	
Version		vSRX Junos 22.4R2.8	
vCPU		8	
Memory		21 GB	
Routing Functionality		Result	
Unrestricted Traffic Test		Pass	
Segmented Traffic Test		Pass	
Access Control		Result	
Simple Policies		Pass	
Complex Multi-Zone Policies		Pass	
TLS/SSL Support			
Cipher Suites	Prevalence	Version	Result
TLS_AES_256_GCM_SHA384 (0x13, 0x02)	66.51%	TLS 1.3	Pass
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)	11.85%	TLS 1.2	Pass
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)	9.26%	TLS 1.2	Pass
TLS_AES_128_GCM_SHA256 (0x13, 0x01)	8.07%	TLS 1.3	Pass
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA8)	1.72%	TLS 1.2	Pass
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28)	0.68%	TLS 1.2	Pass
TLS_CHACHA20_POLY1305_SHA256 (0x13, 0x03)	0.55%	TLS 1.3	Pass
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C)	0.42%	TLS 1.2	Pass
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA9)	0.27%	TLS 1.2	Pass
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B)	0.20%	TLS 1.2	Pass
Null ciphers (no encryption of data)		Version	Result
TLS_RSA_WITH_NULL_MD5 (0x00, 0x01)		SSL 3.0	Pass
TLS_RSA_WITH_NULL_SHA (0x00, 0x02)		SSL 3.0	Pass
Anonymous Ciphers (no authorization)		Version	Result
TLS_DH_anon_WITH_AES_256_CBC_SHA (0x00, 0x3a)		SSL 3.0	Pass
TLS_DH_anon_WITH_RC4_128_MD5 (0x00, 0x18)		SSL 3.0	Pass
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA (0x00, 0x1b)		SSL 3.0	Pass
Decryption Validation			Supported
Decryption Bypass Exceptions			Supported
TLS Session Reuse - Session Tickets			Supported
TLS Session Reuse - Session IDs			Supported

Threat Prevention	
False Positives	Result
False Positives	100%
Exploits	Block Rate
Exploits without Background Network Load	99.70%
Exploits with Background Network Load	99.70%
Evasions IP Packet Fragmentation	Result
Fragment IP packets (1032-byte)	Pass
Fragment IP packets (520-byte)	Pass
Fragment IP packets (72-byte)	Pass
Fragment IP packets (8-byte)	Pass
Fragment IP packets (24-byte); Order packets (reverse)	Pass
Fragment IP packets (24-byte); Order packets (random)	Pass
Fragment IP packets (24-byte); Delay packet (first) (100 milliseconds)	Pass
Fragment IP packets (24-byte); Delay packet (random) (100 milliseconds)	Pass
Fragment IP packets (24-byte); Delay packet (last) (100 milliseconds)	Pass
Fragment IP packets with partial overlap favoring new (24-byte)	Pass
Fragment IP packets (56-byte); Fragment IP packets (24-byte)	Pass
Fragment IP packets (40-byte); Fragment IP packets (16-byte)	Pass
Fragment IP packets (80-byte); Fragment IP packets (56-byte); Fragment IP packets (40-byte); Fragment IP packets (32-byte)	Pass
Fragment IP packets (24-byte); Delay packet (first) (100 milliseconds); Delay packet (last) (100 milliseconds)	Pass
Fragment IP packets (24-byte); Order packets (reverse); Delay packet (first) (100 milliseconds); Delay packet (last) (100 milliseconds)	Pass
Fragment IP packets with partial overlap favoring new (24-byte); Order packets (reverse)	Pass
Fragment IP packets with partial overlap favoring new (24-byte); Order packets (random)	Pass
Fragment IP packets with partial overlap favoring new (24-byte); Delay packet (last) (100 milliseconds)	Pass
JSON Obfuscation	Result
Unicode-Escape JSON Strings	Pass
TCP Transfer Control Block	Result
TCP Split-Handshake	Pass
IP Address Spoofing	Result
Spoof IP Address (127.0.0.1)	Pass
Spoof IP Address (10.0.199.199)	Pass
TCP Stream Segmentation	Result
Segment TCP Segments (1-byte)	Pass
Segment TCP Segments (2-byte)	Pass
Segment TCP Segments (3-byte)	Pass
Segment TCP Segments (4-byte)	Pass
Segment TCP Segments (5-byte)	Pass
Segment TCP Segments (6-byte)	Pass
Segment TCP Segments (7-byte)	Pass
Segment TCP Segments (8-byte)	Pass

Segment TCP Segments (9-byte)	Pass
Segment TCP Segments (15-byte)	Pass
Segment TCP Segments (16-byte)	Pass
Segment TCP Segments (18-byte)	Pass
Segment TCP Segments (31-byte)	Pass
Segment TCP Segments (32-byte)	Pass
Segment TCP Segments (33-byte)	Pass
Segment TCP Segments (34-byte)	Pass
Segment TCP Segments (63-byte)	Pass
Segment TCP Segments (64-byte)	Pass
Segment TCP Segments (65-byte)	Pass
Segment TCP Segments (127-byte)	Pass
Segment TCP Segments (128-byte)	Pass
Segment TCP Segments (129-byte)	Pass
Segment TCP Segments (255-byte)	Pass
Segment TCP Segments (256-byte)	Pass
Segment TCP Segments (257-byte)	Pass
Segment TCP Segments (511-byte)	Pass
Segment TCP Segments (512-byte)	Pass
Segment TCP Segments (513-byte)	Pass
Segment TCP Segments (1023-byte)	Pass
Segment TCP Segments (1024-byte)	Pass
Segment TCP Segments (1025-byte)	Pass
Segment TCP Segments (33-byte); Order packets (reverse)	Pass
Segment TCP Segments (33-byte); Order packets (random)	Pass
Segment TCP Segments (33-byte); Delay packet (first) (100 milliseconds)	Pass
Segment TCP Segments (33-byte); Delay packet (random) (100 milliseconds)	Pass
Segment TCP Segments (33-byte); Delay packet (last) (100 milliseconds)	Pass
Segment TCP Segments (64-byte); Delay packet (first) (100 milliseconds)	Pass
Segment TCP Segments (128-byte); Delay packet (first) (100 milliseconds)	Pass
Segment TCP Segments (255-byte); Delay packet (first) (100 milliseconds)	Pass
Segment TCP Segments (256-byte); Delay packet (first) (100 milliseconds)	Pass
Segment TCP Segments (257-byte); Delay packet (first) (100 milliseconds)	Pass
Segment TCP Segments with partial overlap favoring new (3-byte)	Pass
Segment TCP Segments with partial overlap favoring new (3-byte) then no overlap (1-byte)	Pass
Segment TCP Segments with partial overlap favoring new (3-byte) then partial overlap favoring old (1-byte)	Pass
Segment TCP Segments with partial overlap favoring new (15-byte) then no overlap (3-byte)	Pass
Segment TCP Segments with partial overlap favoring new (15-byte) then partial overlap favoring old (3-byte)	Pass
Segment TCP Segments (7-byte); Segment TCP Segments (4-byte)	Pass
Segment TCP Segments (3-byte); Segment TCP Segments (2-byte)	Pass
Segment TCP Segments (10-byte); Segment TCP Segments (7-byte); Segment TCP Segments (5-byte); Segment TCP Segments (4-byte)	Pass
Segment TCP Segments (33-byte); Delay packet (first) (100 milliseconds); Delay packet (last) (100 milliseconds)	Pass
Segment TCP Segments (33-byte); Order packets (reverse); Delay packet (first) (100 milliseconds); Delay packet (last) (100 milliseconds)	Pass

Segment TCP Segments with partial overlap favoring new (33-byte); Order packets (random)	Pass
HTTP Chunked Encoding	Result
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('00000000') (before)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (' ') (before)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x17') (after)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x1c') (after)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (',') (after)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (' $') (after)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header ('-0')	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0')	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (420-byte)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (257-byte)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (256-byte)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (255-byte)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (16-byte)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (2-byte)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (1-byte)	Pass
No HTTP Content Encoding; HTTP Identity Transfer Encoding	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (7-byte)	Pass
HTTP Compression	Result
HTTP Identity Content Encoding; No HTTP Transfer Encoding	Pass
HTTP Gzip Compression Content Encoding; No HTTP Transfer Encoding	Pass
HTTP Deflate Compression Content Encoding; No HTTP Transfer Encoding	Pass
HTTP Brotli Compression Content Encoding; No HTTP Transfer Encoding	Pass
HTTP Headers	Result
No HTTP Content Encoding; No HTTP Transfer Encoding; Replace the HTTP End of Headers Token with ('\r\n\x10\r\n\r\n')	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Replace the HTTP End of Headers Token with ('\n\r\r\n')	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Replace the HTTP End of Headers Token with ('\n\x06\x11\n\n')	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Replace the HTTP End of Headers Token with ('\n3\n\x03\n\n')	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Add HTTP header ('Transfer-Encoding: chunked,') (after)	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Add HTTP header ('X-Custom-Encoding: chunked') (after); Add HTTP header ('\tTransfer-Encoding: chonked') (after)	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Add HTTP header ('Content-Encoding: gzip pizza') (after)	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Add HTTP header ('Transfer-Encoding: gzip') (after)	Pass

No HTTP Content Encoding; No HTTP Transfer Encoding; Add HTTP header ('Content-Encoding: gzip,') (after)	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Add HTTP header ('Content-Encoding: br pizza') (after)	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Add HTTP header ('Transfer-Encoding: br') (after)	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Add HTTP header ('Content-Encoding: br,') (after)	Pass
No HTTP Content Encoding; No HTTP Transfer Encoding; Add HTTP header ('Content-Encoding: gzip') (after); Add HTTP header ('Content-Encoding: br') (after)	Pass
Layered Evasions	
HTTP Compression, HTTP Chunked Encoding	Result
HTTP Identity Content Encoding; HTTP Chunked Transfer Encoding (7-byte)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (7-byte)	Pass
HTTP Deflate Compression Content Encoding; HTTP Chunked Transfer Encoding (7-byte)	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (7-byte)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('00000000') (before)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (' ') (before)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x17') (after)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x1c') (after)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (',') (after)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (' $') (after)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header ('-0')	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0')	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('00000000') (before)	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (' ') (before)	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after)	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x17') (after)	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x1c') (after)	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (',') (after)	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (' $') (after)	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header ('-0')	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0')	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0')	Pass
HTTP Chunked Encoding, HTTP Headers	Result

No HTTP Content Encoding; HTTP Chunked Transfer Encoding (7-byte); Replace HTTP header ('Transfer-Encoding: chunked') with ('Transfer-Encoding: chunked,')	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (7-byte); Replace HTTP header ('Transfer-Encoding: chunked') with ('\r\rTransfer-Encoding: chunked')	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (7-byte); Replace HTTP header ('Transfer-Encoding: chunked') with ('\t\t\t\tTransfer-Encoding: chunked')	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (7-byte); Add HTTP header ('Content-Encoding: gzip pizza') (after)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (7-byte); Add HTTP header ('Content-Encoding: gzip,') (after)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (7-byte); Add HTTP header ('Content-Encoding: br pizza') (after)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (7-byte); Add HTTP header ('Content-Encoding: br,') (after)	Pass
No HTTP Content Encoding; HTTP Chunked Transfer Encoding (7-byte); Add HTTP header ('Content-Encoding: gzip') (after); Add HTTP header ('Content-Encoding: br') (after)	Pass
HTTP Compression, HTTP Chunked Encoding, TCP Stream Segmentation	Result
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments (3-byte)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (3-byte)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then no overlap (3-byte)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then partial overlap favoring old (3-byte)	Pass
HTTP Compression, HTTP Chunked Encoding, TCP Stream Segmentation, IP Packet Fragmentation	Result
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments (3-byte); Fragment IP packets (16-byte)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (3-byte); Fragment IP packets (16-byte)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then no overlap (3-byte); Fragment IP packets (16-byte)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then partial overlap favoring old (3-byte); Fragment IP packets (16-byte)	Pass
IP Packet Fragmentation, IP Insertion	Result
Fragment IP packets (24-byte); Interleave Chaff IP Packets (Invalid IP option) (before)	Pass
Fragment IP packets (24-byte); Interleave Chaff IP Packets (Invalid IP option) (after)	Pass
Fragment IP packets (24-byte); Interleave Chaff IP Packets (Invalid IP Checksum) (before)	Pass
Fragment IP packets (24-byte); Interleave Chaff IP Packets (Invalid IP Checksum) (after)	Pass
Fragment IP packets (24-byte); Interleave Chaff IP Packets (Invalid IP option) (before and after)	Pass
Fragment IP packets (24-byte); Interleave Chaff IP Packets (Invalid IP Checksum) (before and after)	Pass

Fragment IP packets with partial overlap favoring new (24-byte); Interleave Chaff IP Packets (Invalid IP option) (before and after)	Pass
Fragment IP packets with partial overlap favoring new (24-byte); Interleave Chaff IP Packets (Invalid IP Checksum) (before and after)	Pass
Fragment IP packets with partial overlap favoring new (24-byte); Delay packet (last) (100 milliseconds); Interleave Chaff IP Packets (Invalid IP option) (before and after)	Pass
Fragment IP packets with partial overlap favoring new (24-byte); Delay packet (last) (100 milliseconds); Interleave Chaff IP Packets (Invalid IP Checksum) (before and after)	Pass
Fragment IP packets with partial overlap favoring new (24-byte); Delay packet (last) (100 milliseconds); Interleave Chaff IP Packets (Invalid IP Checksum) (before and after); Delay packet (last) (100 milliseconds)	Pass
JSON Obfuscation, HTTP Compression, HTTP Chunked Encoding, TCP Stream Segmentation	Result
Unicode-Escape JSON Strings; HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments (3-byte)	Pass
Unicode-Escape JSON Strings; HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (3-byte)	Pass
Unicode-Escape JSON Strings; HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then no overlap (3-byte)	Pass
Unicode-Escape JSON Strings; HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then partial overlap favoring old (3-byte)	Pass
Unicode-Escape JSON Strings; HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments (3-byte)	Result
Unicode-Escape JSON Strings; HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (3-byte)	Pass
Unicode-Escape JSON Strings; HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then no overlap (3-byte)	Pass
Unicode-Escape JSON Strings; HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then partial overlap favoring old (3-byte)	Pass
JSON Obfuscation, HTTP Compression, HTTP Chunked Encoding, TCP Stream Segmentation, IP Packet Fragmentation	Result
Unicode-Escape JSON Strings; HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments (3-byte); Fragment IP packets (16-byte)	Pass
Unicode-Escape JSON Strings; HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (3-byte); Fragment IP packets (16-byte)	Pass

Unicode-Escape JSON Strings; HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then no overlap (3-byte); Fragment IP packets (16-byte)	Pass
Unicode-Escape JSON Strings; HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then partial overlap favoring old (3-byte); Fragment IP packets (16-byte)	Pass
Unicode-Escape JSON Strings; HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments (3-byte); Fragment IP packets (16-byte)	Pass
Unicode-Escape JSON Strings; HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (3-byte); Fragment IP packets (16-byte)	Pass
Unicode-Escape JSON Strings; HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then no overlap (3-byte); Fragment IP packets (16-byte)	Pass
Unicode-Escape JSON Strings; HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (256-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers ('\x04') (after); Replace the Chunk Size in the Terminal HTTP Chunk Header ('0.0'); Segment TCP Segments with partial overlap favoring new (5-byte) then partial overlap favoring old (3-byte); Fragment IP packets (16-byte)	Pass
TCP Stream Segmentation, IP Packet Fragmentation	Result
Segment TCP Segments (3-byte); Fragment IP packets (16-byte)	Pass
Segment TCP Segments with partial overlap favoring new (3-byte); Fragment IP packets (16-byte)	Pass
Segment TCP Segments with partial overlap favoring new (5-byte) then no overlap (3-byte); Fragment IP packets (16-byte)	Pass
Segment TCP Segments with partial overlap favoring new (5-byte) then partial overlap favoring old (3-byte); Fragment IP packets (16-byte)	Pass
TCP Stream Segmentation, TCP Insertion	Result
Segment TCP Segments (3-byte); Interleave Chaff TCP Segments (Invalid TCP Checksum) (before)	Pass
Segment TCP Segments (3-byte); Interleave Chaff TCP Segments (Invalid TCP Checksum) (after)	Pass
Segment TCP Segments (3-byte); Interleave Chaff TCP Segments (Older PAWS Timestamps) (before)	Pass
Segment TCP Segments (3-byte); Interleave Chaff TCP Segments (Older PAWS Timestamps) (after)	Pass
Segment TCP Segments (33-byte); Interleave Chaff TCP Segments (Invalid TCP Checksum) (sandwich)	Pass
Segment TCP Segments (33-byte); Interleave Chaff TCP Segments (Older PAWS Timestamps) (before)	Pass
Segment TCP Segments with partial overlap favoring new (33-byte); Interleave Chaff TCP Segments (Invalid TCP Checksum) (sandwich)	Pass
Segment TCP Segments with partial overlap favoring new (33-byte); Interleave Chaff TCP Segments (Older PAWS Timestamps) (after)	Pass
TCP Stream Segmentation, TCP Transfer Control Block	Result
Segment TCP Segments (3-byte); Interleave Chaff TCP Segments (Requests to Resync Sequence Numbers Mid-stream) (before)	Pass
Segment TCP Segments (3-byte); Interleave Chaff TCP Segments (Requests to Resync Sequence Numbers Mid-stream) (after)	Pass
Segment TCP Segments (3-byte); Interleave Chaff TCP Segments (Out-Of-Window Sequence Numbers) (before)	Pass
Segment TCP Segments (3-byte); Interleave Chaff TCP Segments (Out-Of-Window Sequence Numbers) (after)	Pass
Segment TCP Segments (33-byte); Interleave Chaff TCP Segments (Out-Of-Window Sequence Numbers) (sandwich)	Pass

Segment TCP Segments (33-byte); Interleave Chaff TCP Segments (Requests to Resync Sequence Numbers Mid-stream) (sandwich)	Pass
Segment TCP Segments with partial overlap favoring new (33-byte); Interleave Chaff TCP Segments (Out-Of-Window Sequence Numbers) (sandwich)	Pass
Segment TCP Segments with partial overlap favoring new (33-byte); Interleave Chaff TCP Segments (Requests to Resync Sequence Numbers Mid-stream) (sandwich)	Pass

Performance		
Raw Packet Processing Performance (UDP Throughput)	Throughput (Mbps)	Latency (µs)
64 Byte Frames	1,297	0.02
128 Byte Frames	2,260	0.01
256 Byte Frames	4,211	0.01
512 Byte Frames	7,568	0.01
1024 Byte Frames	9,977	0.01
1280 Byte Frames	10,000	0.01
1400 Byte Frames	10,000	0.01

Maximum Capacity	CPS	TPS	
Max Concurrent TCP Connection	63,403	-	
Max TCP TPS	-	39,070	
HTTP Capacity (without transaction delay)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 115.6 KB Response	3,085	3,085	739.73
2,000 Connections Per Second - 57.4 KB Response	4,800	2,400	200.54
4,000 Connections Per Second - 28.0 KB Response	7,468	1,867	154.20
8,000 Connections Per Second - 13.5 KB Response	11,085	1,386	76.12
16,000 Connections Per Second - 6.4 KB Response	13,700	856	52.22
32,000 Connections Per Second - 2.7 KB Response	15,100	472	16.80
HTTP Capacity (with transaction delay)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 115.6 KB Response	2,873	2,873	10040.48
2,000 Connections Per Second - 57.4 KB Response	4,289	2,145	10020.91
4,000 Connections Per Second - 28.0 KB Response	7,247	1,812	10013.42
8,000 Connections Per Second - 13.5 KB Response	10,610	1,326	10018.78
16,000 Connections Per Second - 6.4 KB Response	12,000	750	10010.47
32,000 Connections Per Second - 2.7 KB Response	14,980	468	10005.31
HTTPS Capacity (0x13, 0x02)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 113.8 KB Response	2,137	2,137	51.72
2,000 Connections Per Second - 54.9 KB Response	2,502	1,251	14.20
4,000 Connections Per Second - 25.7 KB Response	2,632	658	10.14
8,000 Connections Per Second - 11.2 KB Response	2,774	347	8.78
16,000 Connections Per Second - 3.9 KB Response	2,803	175	2.71
32,000 Connections Per Second - 0.2 KB Response	2,827	88	0.10
HTTPS Capacity (0xC0, 0x30)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 115.0 KB Response	2,020	2,020	39.06
2,000 Connections Per Second - 56.3 KB Response	2,515	1,258	16.33
4,000 Connections Per Second - 27.0 KB Response	2,778	695	9.41

8,000 Connections Per Second - 12.3 KB Response	2,802	350	2.89
16,000 Connections Per Second - 5.0 KB Response	2,850	178	1.74
32,000 Connections Per Second - 1.4 KB Response	3,020	94	0.10
HTTPS Capacity (0xC0, 0x2F)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 115.0 KB Response	2,086	2,086	103.63
2,000 Connections Per Second - 56.3 KB Response	2,440	1,220	18.06
4,000 Connections Per Second - 27.0 KB Response	2,587	647	7.64
8,000 Connections Per Second - 12.3 KB Response	2,673	334	2.42
16,000 Connections Per Second - 5.0 KB Response	2,906	182	0.49
32,000 Connections Per Second - 1.4 KB Response	3,020	94	0.10
HTTPS Capacity (0x13, 0x01)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 113.8 KB Response	2,329	2,329	97.19
2,000 Connections Per Second - 54.9 KB Response	2,503	1,252	38.29
4,000 Connections Per Second - 25.7 KB Response	2,660	665	10.55
8,000 Connections Per Second - 11.2 KB Response	2,862	358	3.30
16,000 Connections Per Second - 3.9 KB Response	2,927	183	2.38
32,000 Connections Per Second - 0.2 KB Response	3,044	95	0.10
Delta Capacity			
HTTP Capacity (without transaction delay)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 113.8 KB Response	3,197	3,197	240.65
2,000 Connections Per Second - 54.9 KB Response	4,804	2,402	204.47
4,000 Connections Per Second - 25.7 KB Response	7,996	1,999	114.87
8,000 Connections Per Second - 11.2 KB Response	11,747	1,468	77.67
16,000 Connections Per Second - 3.9 KB Response	14,801	925	36.67
32,000 Connections Per Second - 0.2 KB Response	18,241	570	0.10
HTTPS Capacity (0x13, 0x02)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 113.8 KB Response	2,137	2,137	51.72
2,000 Connections Per Second - 54.9 KB Response	2,502	1,251	14.20
4,000 Connections Per Second - 25.7 KB Response	2,632	658	10.14
8,000 Connections Per Second - 11.2 KB Response	2,774	347	8.78
16,000 Connections Per Second - 3.9 KB Response	2,803	175	2.71
32,000 Connections Per Second - 0.2 KB Response	2,827	88	0.10
Stability and Reliability		Result	
Protocol Fuzzing & Mutation		Pass	
Blocking with Minimal Load		Pass	
Blocking Under Load		Pass	
Attack Detection/Blocking – Normal Load		Pass	
State Preservation – Normal Load		Pass	
Pass Legitimate Traffic – Normal Load		Pass	
State Preservation – Maximum Exceeded		Pass	
Drop Traffic – Maximum Exceeded		Pass	

Appendix B – CyberRatings Rating Matrix

Rating	Definition
Recommended	A product with the "*Recommended*" rating has the highest rating assigned by CyberRatings. These products are recommended for *security, performance,* and *value.* The product's capacity to meet its commitments to consumers is extremely strong.
Neutral	A "*Neutral*" product is less capable than the higher-rated categories. These devices would be suitable for environments where budget is a priority, and a slightly lower level of protection is acceptable in exchange for a lower cost of ownership. The product's capacity to meet its commitments to consumers is still strong.
Caution	A product rated "*Caution"* offers poor value for money given the measured security effectiveness, performance, and 3-year cost. Products that earn a *Caution* rating from CyberRatings should not be short-listed or renewed.

SPECIAL THANKS

We would like to issue a special thank you to Keysight for providing their CyPerf and Breaking Point tools for us to test the performance, TLS functionality, and stability of Cloud Network Firewalls.

We would also like to thank TeraPackets for providing us with their Threat Replayer tool which enabled us to accurately replay exploits in a cloud environment.

AUTHORS

Thomas Skybakmoen, Ahmed Basheer, Vikram Phatak

CONTACT INFORMATION

CyberRatings.org
2303 Ranch Road 620 South
Suite 160, #501
Austin, TX 78734
info@cyberratings.org
www.cyberratings.org