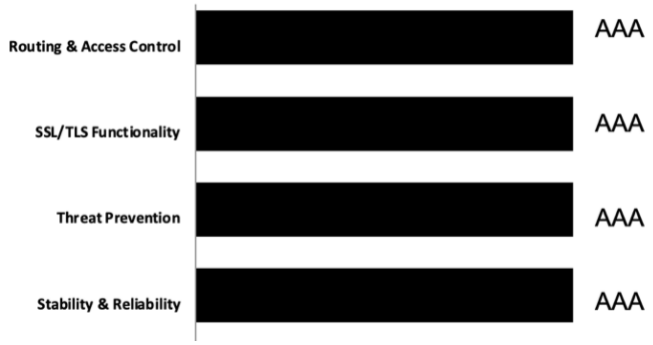
Forcepoint 2205

RECOMMENDED

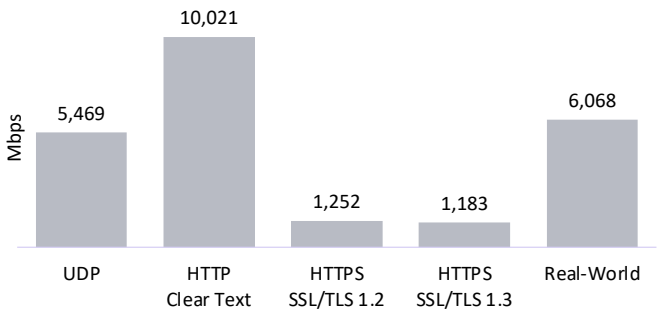
OVERVIEW

In Q2 2023, CyberRatings.org performed an independent test of the Forcepoint 2205 NGFW version 7.0.1.28052 against the Enterprise Firewall Test Methodology v3.0 at our facility in Austin, Texas. The product was subjected to thorough testing to determine how it handled TLS/SSL 1.2 and 1.3 cipher suites, how it defended against 1,724 exploits, whether its protection could be bypassed by any of 1,482 evasions, and whether 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.

99.48% SECURITY EFFECTIVENESS



RATED THROUGHPUT – 4,235 MBPS



Max Concurrent TCP Connection CPS	689,996
Max TCP CPS	54,380
Max HTTP CPS	53,280
Max HTTP TPS	80,630
Max HTTPS CPS (0x13, 0x01)	2,214
Max HTTPS CPS (0x13, 0x02)	4,150
Max HTTPS CPS (0xC0, 0x2F)	3,389
Max HTTPS CPS (0xC0, 0x30)	4,374

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

THREAT PREVENTION

Client-Initiated Exploits	680/681
Server-Initiated Exploits	1035/1043
Client-Initiated Evasions	753/753
Server-Initiated Evasions	729/729

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

MSRP + SUPPORT & MAINTENANCE

3-Year Cost	$32,915

Routing & Access Control AAA

ACCESS CONTROL

Throughout its history, the goal of a firewall has been to enforce an access control policy between two networks. Rules are 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.

Routing Functionality	Results
Unrestricted Traffic Test	Pass
Segmented Traffic Test	Pass

Figure 1 – Routing Functionality

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. Traffic was tested at each level of complexity to ensure specified policies were enforced.

Access Control	Results
Simple Policies	Pass
Complex Multi-Zone Policies	Pass

Figure 2 – Access Control

SSL/TLS Functionality AAA

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 January 2023, over 77% of web traffic is being sent over HTTPS.[1]

While CyberRatings believes using encryption is good, SSL/TLS 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 enterprise firewalls were tested to provide visibility into the SSL/TLS payloads and detect attacks concealed by encryption as well as attacks against the encryption protocols themselves. The table below lists the tested SSL/TLS in order of prevalence[2] per March 2023.

DECRYPTION VALIDATION

Version	Prevalence	Cipher Suites	Results
TLS 1.3	63.90%	TLS_AES_256_GCM_SHA384 (0x13, 0x02)	Pass
TLS 1.2	13.70%	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)	Pass
TLS 1.2	9.90%	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)	Pass
TLS 1.3	7.70%	TLS_AES_128_GCM_SHA256 (0x13, 0x01)	Pass
TLS 1.2	1.30%	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA8)	Pass
TLS 1.2	1.10%	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28)	Pass
TLS 1.3	1.10%	TLS_CHACHA20_POLY1305_SHA256 (0x13, 0x03)	Pass
TLS 1.2	0.30%	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA9)	Pass
TLS 1.2	0.30%	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C)	Pass
TLS 1.2	0.20%	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B)	Pass

Figure 3 – SSL/TLS 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 SSL/TLS 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. Lastly, we tested TLS session reuse; to improve performance and reduce the overhead associated with conducting the full handshake for each session. The TLS protocol allows for abbreviated handshakes, which reuse previously established sessions.

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 AAA

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. Exploit sets for individual tests are selected based on CVSS score (how widely used is an application + what can an attacker do?), use case, and customer relevance. This has implications for the age of exploits since some applications in industrial environments are deployed and then left untouched for years. In contrast, other applications within office environments are refreshed every 5-7 years.

EXPLOIT PROTECTION

99.48% BLOCKED (1,715/1724)

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.

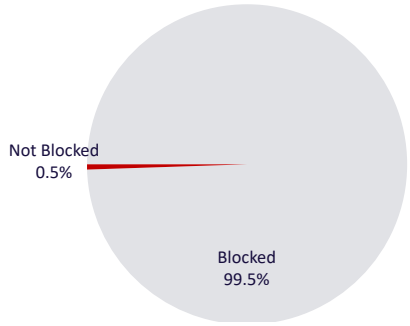


Figure 4 – Exploit Block Rate

Coverage by Attack Vector

Because a failure to block attacks could result in significant compromise and could severely impact critical business systems, firewalls should be evaluated against a broad set of exploits. Exploits can be categorized as either client-initiated or server-initiated. Server-initiated exploits are threats executed remotely against a vulnerable application and/or operating system by an individual, while client-initiated exploits are initiated by the vulnerable target. Client-initiated exploits are the most common type of attack experienced by the end user, and the attacker has little or no control as to when the threat is executed.

Attack Vector	Result
Client-Initiated	680/681
Server-Initiated	1035/1043

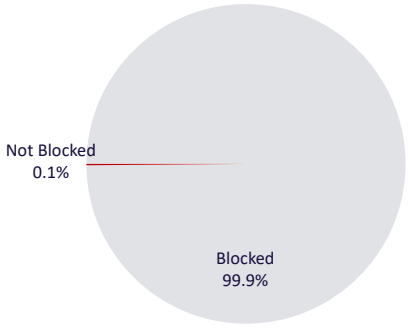


Figure 5 – Exploit Block Rate (Client-Initiated)

False Positives

A key to effective protection is the ability to correctly identify and allow legitimate traffic while maintaining protection against malware, exploits, and phishing attacks. False positives are any legitimate, non-malicious content/traffic perceived as malicious. False positive tests flex 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.

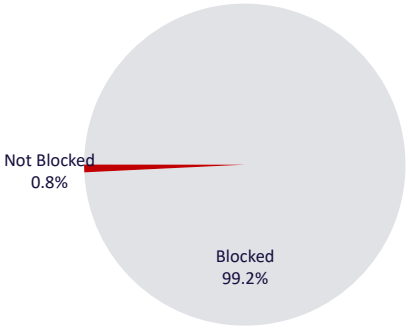


Figure 6 – Exploit Block Rate (Server-Initiated)

COVERAGE BY DATE

Figure 5 provides insight into whether or not a vendor is aging out protection signatures aggressively enough to preserve performance levels. It also reveals whether a product lags behind in protection for the most current vulnerabilities. CyberRatings reports exploits by individual years for the past ten+ years.

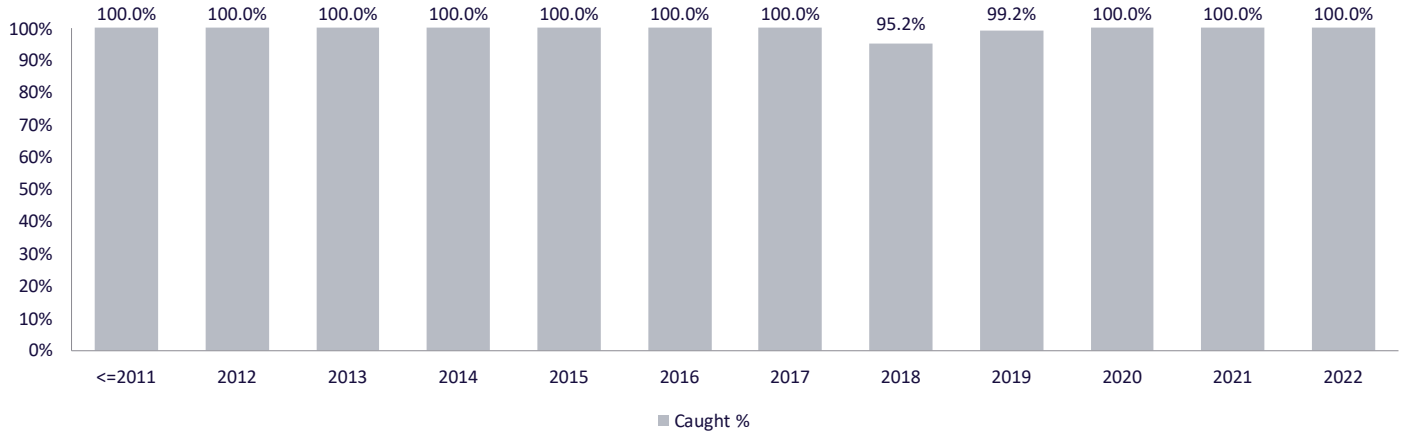


Figure 7 – Coverage by Date

COVERAGE BY TARGET VENDOR

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

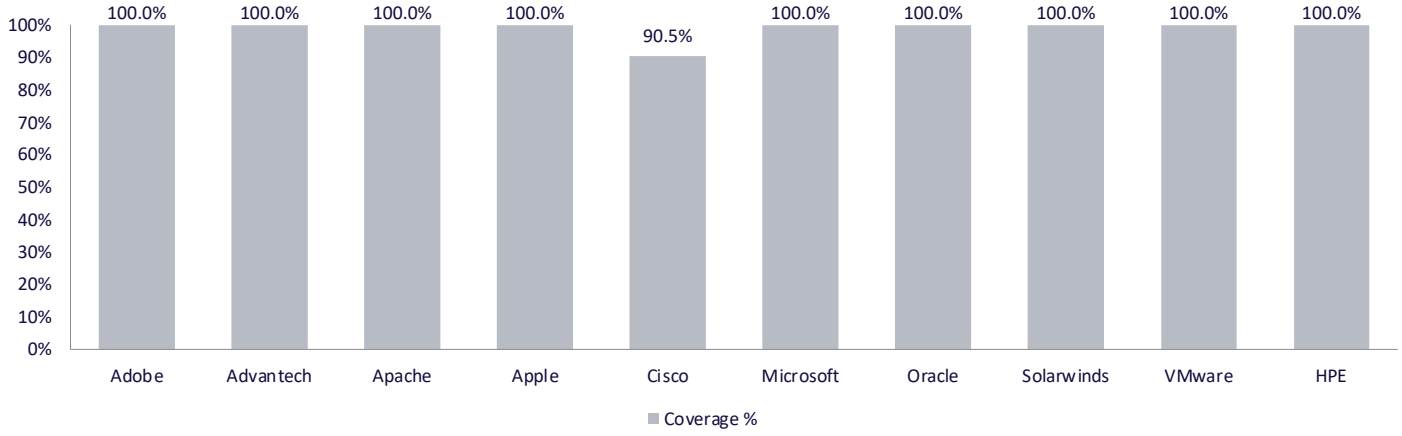


Figure 8 – Coverage for Top Vendors

RESISTANCE TO EVASIONS

100.00% EFFECTIVE (1,482/1,482)

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

Handling evasions is hard. And to our knowledge, this was the most comprehensive evasion test performed to date. 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.

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 HTML evasion is limited to far fewer exploits.

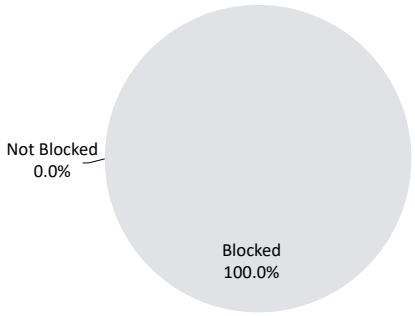


Figure 9 – Evasion Effectiveness

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	Number of Evasions Tested	Number of Evasions Blocked
Client-initiated evasions		
IP Spoofing	5	5
IP Fragmentation	96	96
TCP Segmentation	264	264
Layered Evasions	16	16
HTTP Obfuscation	172	172
HTTP Compression	72	72
HTML Obfuscation	124	124
Layered Evasions	4	4
Server-initiated evasions		
TCP Split Handshake	5	5
IP Fragmentation	238	238
TCP Segmentation	458	458
Layered Evasions	28	28

Figure 10 – Evasions by Technique

Evasions	Result
IP Address Spoofing	**Pass**
TCP Split Handshake Spoofing	**Pass**
IP Packet Fragmentation (Client & Server)	**Result**
(small IP fragments (32 bytes))	Pass
(small IP fragments (24 bytes))	Pass
(small IP fragments (16 bytes))	Pass
(small IP fragments (8 bytes))	Pass
(small IP fragments in order)	Pass
(small IP fragments in reverse order)	Pass
(small IP fragments in random order)	Pass
(small IP fragments; delay first fragment)	Pass
(small IP fragments; delay random fragment)	Pass
(small IP fragments; delay last fragment)	Pass
(overlapping small IP fragments)	Pass
(small IP fragments; interleave chaff (duplicate))	Pass
(small IP fragments; interleave chaff (duplicate with dummy payload) after fragment)	Pass
(small IP fragments; interleave chaff (invalid IP options) before fragment)	Pass
(small IP fragments; interleave chaff (invalid IP options) after fragment)	Pass
(small IP fragments; interleave chaff (invalid IP checksum) before fragment)	Pass
(small IP fragments; interleave chaff (invalid IP checksum) after fragment)	Pass
(small IP fragments (24-32 bytes))	Pass
(small IP fragments (16-24 bytes))	Pass
(small IP fragments (8-32 bytes))	Pass
(small IP fragments; delay first fragment; delay last fragment)	Pass
(small IP fragments in reverse order; delay first fragment; delay last fragment)	Pass
(small IP fragments; interleave chaff (invalid IP options) before and after fragment)	Pass
(small IP fragments; interleave chaff (invalid IP checksum) before and after fragment)	Pass
(overlapping small IP fragments favoring new data)	Pass
(small IP fragments; interleave chaff (duplicate with dummy payload) before fragment)	Pass
(overlapping small IP fragments favoring new data in reverse order)	Pass
(overlapping small IP fragments favoring new data in random order)	Pass
(overlapping small IP fragments favoring new data; interleave chaff (invalid IP options))	Pass
(overlapping small IP fragments favoring new data; interleave chaff (invalid IP checksum))	Pass
(overlapping small IP fragments favoring new data in reverse order; delay last fragment	Pass
(overlapping small IP fragments favoring new data in reverse order; delay last fragment; interleave chaff (invalid IP options))	Pass
(overlapping small IP fragments favoring new data in reverse order; delay last fragment; interleave chaff (invalid IP checksum))	Pass
(overlapping small IP fragments favoring new data in reverse order; delay last fragment; interleave chaff (invalid IP options); interleave chaff (invalid IP checksum))	Pass
(overlapping small IP fragments favoring new data in reverse order; delay last fragment; interleave chaff (invalid IP options); interleave chaff (invalid IP checksum); delay first fragment)	Pass

TCP Segmentation (Client & Server)	Result
(small TCP segments (1025 bytes))	Pass
(small TCP segments (1024 bytes))	Pass
(small TCP segments (1023 bytes))	Pass
(small TCP segments (513 bytes))	Pass
(small TCP segments (512 bytes))	Pass
(small TCP segments (511 bytes))	Pass
(small TCP segments (257 bytes))	Pass
(small TCP segments (256 bytes))	Pass
(small TCP segments (255 bytes))	Pass
(small TCP segments (129 bytes))	Pass
(small TCP segments (128 bytes))	Pass
(small TCP segments (127 bytes))	Pass
(small TCP segments (65 bytes))	Pass
(small TCP segments (64 bytes))	Pass
(small TCP segments (63 bytes))	Pass
(small TCP segments (33 bytes))	Pass
(small TCP segments (32 bytes))	Pass
(small TCP segments (31 bytes))	Pass
(small TCP segments (17 bytes))	Pass
(small TCP segments (16 bytes))	Pass
(small TCP segments (15 bytes))	Pass
(small TCP segments (9 bytes))	Pass
(small TCP segments (8 bytes))	Pass
(small TCP segments (7 bytes))	Pass
(small TCP segments (5 bytes))	Pass
(small TCP segments (4 bytes))	Pass
(small TCP segments (3 bytes))	Pass
(small TCP segments (2 bytes))	Pass
(small TCP segments (1 bytes))	Pass
(small TCP segments in order)	Pass
(small TCP segments in reverse order)	Pass
(small TCP segments in random order)	Pass
(small TCP segments (257 bytes); delay first segment)	Pass
(small TCP segments (256 bytes); delay first segment)	Pass
(small TCP segments (128 bytes); delay first segment)	Pass
(small TCP segments (64 bytes); delay first segment)	Pass
(small TCP segments; delay first segment)	Pass
(small TCP segments; delay random segment)	Pass
(small TCP segments; delay last segment)	Pass
(overlapping small TCP segments (16 bytes) favoring new data then non-overlapping small TCP segments (2 bytes))	Pass

(overlapping small TCP segments (16 bytes) favoring new data then overlapping small TCP segments (2 bytes) favoring old data)	Pass
(overlapping small TCP segments)	Pass
(overlapping small TCP segments favoring old data)	Pass
(overlapping small TCP segments favoring new data then non-overlapping small TCP segments)	Pass
(overlapping small TCP segments favoring new data then overlapping small TCP segments favoring old data)	Pass
(small TCP segments; interleave chaff (duplicate))	Pass
(small TCP segments; interleave chaff (invalid TCP checksums) before segment)	Pass
(small TCP segments; interleave chaff (invalid TCP checksums) after segment)	Pass
(small TCP segments; interleave chaff (older PAWS timestamps) after segment)	Pass
(small TCP segments; interleave chaff (out-of-window sequence numbers) before segment)	Pass
(small TCP segments; interleave chaff (out-of-window sequence numbers) after segment)	Pass
(small TCP segments; interleave chaff (requests to resynch sequence numbers mid-stream) before segment)	Pass
(small TCP segments; interleave chaff (requests to resynch sequence numbers mid-stream) after segment)	Pass
(small TCP segments (3-4 bytes))	Pass
(small TCP segments (2-3 bytes))	Pass
(small TCP segments (1-4 bytes))	Pass
(small TCP segments; delay first segment; delay last segment)	Pass
(small TCP segments in reverse order; delay first segment; delay last segment)	Pass
(small TCP segments; interleave chaff (invalid TCP checksums) before and after segment)	Pass
(small TCP segments; interleave chaff (out-of-window sequence numbers) before and after segment)	Pass
(small TCP segments; interleave chaff (requests to resynch sequence numbers mid-stream) before and after segment)	Pass
(overlapping small TCP segments favoring old data in random order)	Pass
(overlapping small TCP segments favoring old data; interleave chaff (invalid TCP checksums))	Pass
(overlapping small TCP segments favoring old data; interleave chaff (out-of-window sequence numbers))	Pass
(overlapping small TCP segments favoring old data; interleave chaff (requests to resynch sequence numbers mid-stream))	Pass
(overlapping small TCP segments favoring new data)	Pass
(small TCP segments; interleave chaff (older PAWS timestamps) before segment)	Pass
(overlapping small TCP segments favoring new data in random order)	Pass
(overlapping small TCP segments favoring new data; interleave chaff (invalid TCP checksums))	Pass
(overlapping small TCP segments favoring new data; interleave chaff (older PAWS timestamps))	Pass
(overlapping small TCP segments favoring new data; interleave chaff (out-of-window sequence numbers))	Pass
(overlapping small TCP segments favoring new data; interleave chaff (requests to resynch sequence numbers mid-stream))	Pass
Layered Evasions (Client & Server)	**Result**
(small TCP segments; small IP fragments)	Pass
(overlapping small TCP segments favoring old data; small IP fragments)	Pass
(overlapping small TCP segments favoring new data then non-overlapping small TCP segments ; small IP fragments)	Pass
(overlapping small TCP segments favoring new data then overlapping small TCP segments favoring old data ; small IP fragments)	Pass
(overlapping small TCP segments favoring new data; small IP fragments)	Pass
HTTP Obfuscation	**Result**
(HTTP/0.9 response (no response headers))	Pass
(Declared HTTP/0.9 response; but includes response headers; chunking declared but served without chunking)	Pass

(HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'))	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission (hex '04'))	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission block (hex '17'))	Pass
(HTTP/1.1 chunked response with chunk sizes followed by file separator (hex '1c'))	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a comma (hex '2c'))	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a $ (hex '24'))	Pass
(HTTP/1.1 chunked response with final chunk size of '00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000' (rather than '0'))	Pass
(HTTP/1.1 response with line folded transfer-encoding header declaring chunking ('Transfer-Encoding: ' followed by CRLF (hex '0d 0a') followed by 'chunked' followed by CRLF (hex '0d 0a'); served without chunking)	Pass
(HTTP/1.1 response with transfer-encoding header declaring chunking with lots of whitespace ('Transfer- Encoding:' followed by 8000 spaces (hex '20' * 8000) followed by 'chunked' followed by CRLF (hex '0d 0a'); served chunked)	Pass
(HTTP/1.0 response declaring chunking; served without chunking)	Pass
(HTTP/1.1 response with "Transfer-Encoding: chunked(hex 2C)"; served without chunking)	Pass
(HTTP/1.1 response with "Content-Encoding: gzip(hex 2C)"; served uncompressed)	Pass
(HTTP/1.0 response declaring chunking with invalid content-length header; served without chunking)	Pass
(HTTP/1.1 response with "\tTransfer-Encoding: chunked"; served chunked)	Pass
(HTTP/1.1 response with "\tTransfer-Encoding: chonked" after custom header line with "chunked" as value; served without chunking)	Pass
(HTTP/1.1 response with header with no field name and colon+junk string; followed by '\tTransfer-Encoding: chunked' header; followed by custom header; served chunked)	Pass
(HTTP/1.1 response with "\r\rTransfer-Encoding: chunked"; served chunked)	Pass
(HTTP/1.1\nTransfer-Encoding:chunked; header end \n\n; served chunked)	Pass
(HTTP/1.1 response with "SIP/2.0 200 OK\r\n" before status header; chunked)	Pass
(HTTP/1.1 response with space+junk string followed by \r\n before first header; chunked)	Pass
(HTTP/1.1 response with junk string before status header; chunked)	Pass
(HTTP/1.1 response with header end \n\004\n\n; chunked)	Pass
(HTTP/1.1 response with header end \r\n\010\r\n\r\n; chunked)	Pass
(HTTP/1.1 response with header end \n\r\r\n; chunked)	Pass
(HTTP/1.1 response with header end \n\006\011\n\n; chunked)	Pass
(HTTP/1.1 response with header end \n\033\n\003\n\n; chunked)	Pass
(HTTP/1.1 response with status code 202; with message-body; chunked)	Pass
(HTTP/1.1 response with status code 429; with message-body; chunked)	Pass
(HTTP/1.1 response with status code 300; with message-body; chunked)	Pass
(HTTP/1.1 response with status code 306; with message-body; chunked)	Pass
(HTTP/1.1 response with status code 414; with message-body; chunked)	Pass
(HTTP/1.1 chunked response with no status indicated)	Pass
(No status line; chunking indicated; served unchunked)	Pass
(HTTP/1.1 response with invalid content-length header size declaration followed by space and null (hex '20 00'))	Pass
(HTTP/2.0 declared; served chunked)	Pass
(HTTP/0001.1 declared; served chunked)	Pass
(HTTP/6.-66 declared; served chunked)	Pass
(HTTP/7.7 declared; served chunked)	Pass
(Double Transfer-Encoding: first empty; last chunked. Served with invalid content-length; not chunked.)	Pass

(Relevant headers padded by preceding with hundreds of random custom headers)	Pass
HTTP Compression	**Result**
(HTTP/1.1 response with content-encoding declaration of gzip followed by space+junk string; served uncompressed and chunked)	Pass
(HTTP/1.1 response with content-encoding header for deflate; followed by content-encoding header for gzip; served uncompressed and chunked)	Pass
(HTTP/1.1 response compressed with deflate)	Pass
(HTTP/1.1 response declaring deflate followed by junk string; served uncompressed)	Pass
(HTTP/1.1 response compressed with gzip)	Pass
(HTTP/1.1 response declaring gzip followed by junk string; served uncompressed)	Pass
(HTTP/1.1 response with "Transfer-Encoding: gzip"; served uncompressed)	Pass
(HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission (hex '04'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission block (hex '17'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by file separator (hex '1c'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a comma (hex '2c'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a $ (hex '24'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'); compressed with deflate)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission (hex '04'); compressed with deflate)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission block (hex '17'); compressed with deflate)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by file separator (hex '1c'); compressed with deflate)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a comma (hex '2c'); compressed with deflate)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a $ (hex '24'); compressed with deflate)	Pass
HTML Obfuscation	**Result**
(UTF-8 encoding)	Pass
(UTF-8 encoding with BOM)	Pass
(UTF-16 encoding with BOM)	Pass
(UTF-8 encoding; no http or html declarations)	Pass
(UTF-8 encoding with BOM; no http or html declarations)	Pass
(UTF-16 encoding with BOM; no http or html declarations)	Pass
(UTF-16-LE encoding)	Pass
(UTF-16-BE encoding)	Pass
(UTF-16-LE encoding; no http or html declarations)	Pass
(UTF-16-BE encoding; no http or html declarations)	Pass
(UTF-7 encoding)	Pass
(UTF-7 encoding; no http or html declarations)	Pass
(EICAR string included at top of HTML)	Pass
(padded with <5MB)	Pass
(padded with >5MB and <25MB)	Pass
(padded with >25MB)	Pass
(padded with >5MB and chunked)	Pass
(padded with >5MB and <25MB and chunked)	Pass
(padded with >25MB and chunked)	Pass

(padded with 5MB and compressed with gzip)	Pass
(padded with >5MB and <25MB and compressed with gzip)	Pass
(padded with >25MB and compressed with gzip)	Pass
(padded with 5MB and compressed with deflate)	Pass
(padded with >5MB and <25MB and compressed with deflate)	Pass
(padded with >25MB and compressed with deflate)	Pass
(UTF-8 encoding; padded with >25MB and chunked)	Pass
(UTF-8 encoding with BOM; padded with >25MB and chunked)	Pass
(UTF-16 encoding with BOM; padded with >25MB and chunked)	Pass
(UTF-8 encoding; no http or html declarations; padded with >25MB and chunked)	Pass
(UTF-8 encoding with BOM; no http or html declarations; padded with >25MB and chunked)	Pass
(UTF-16 encoding with BOM; no http or html declarations; padded with >25MB and chunked)	Pass
Combination	**Result**
(UTF-8 encoding; padded with >25MB and chunked; small TCP segments; small IP fragments)	Pass

Figure 11 – Evasions in Detail

Performance

The performance of the enterprise 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.

RATED THROUGHPUT

We measured performance with different packet sizes and payloads to capture the firewall's performance curves for UDP, HTTP, and HTTPS. The "Rated Throughput" is an average of UDP, HTTP, and HTTPS Capacity (1,000, 2,000,4,000, and 8,000 CPS), and the "Real World Application Flows" is a good benchmark for what an enterprise can expect the firewall to achieve in a typical enterprise network.

Performance (Mbps)	
Rated Throughput	4,235 Mbps

Figure 12 – Rated Throughput

RAW PACKET PROCESSING PERFORMANCE (UDP THROUGHPUT)

This test used UDP packets of varying sizes generated by traffic generation appliances. A constant stream of the appropriate packet size — with variable source and destination IP addresses transmitting from a fixed source port to a fixed destination port — was transmitted bidirectionally through each port pair. Each packet contained dummy data and was targeted at a valid port on a valid IP address on the target subnet. The percentage load and frames per second (fps) figures across each inline port pair were verified by network monitoring tools before each test began. Multiple tests were run, and averages were taken where necessary.

This traffic did not attempt to simulate any form of real-world network condition. Therefore, no TCP sessions were created during this test, and there was very little for the detection engine to do.

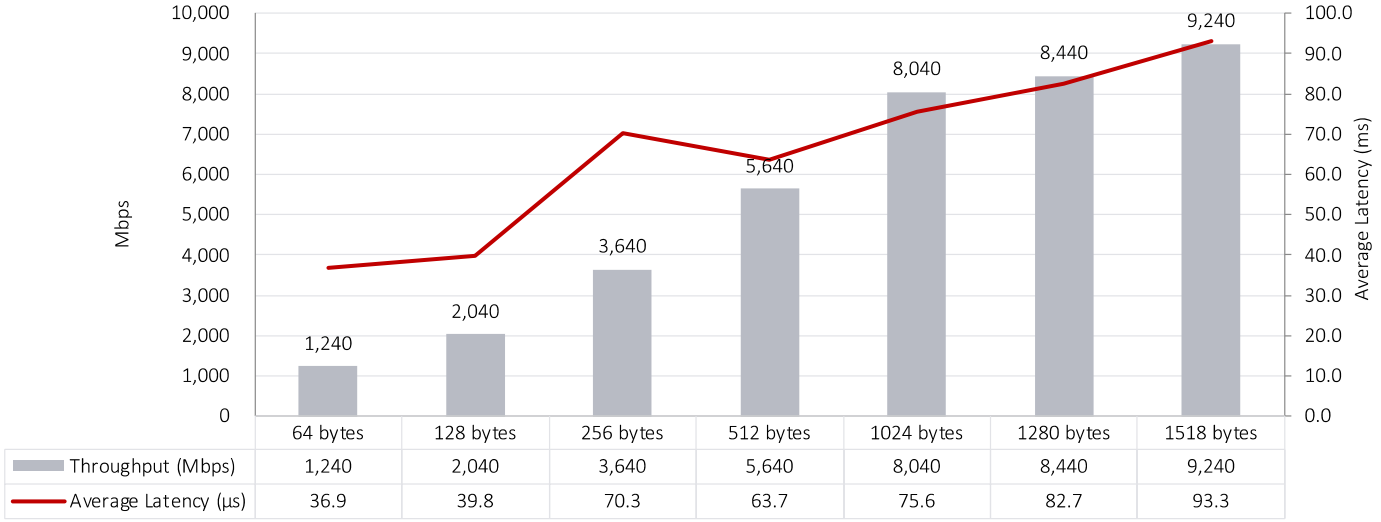
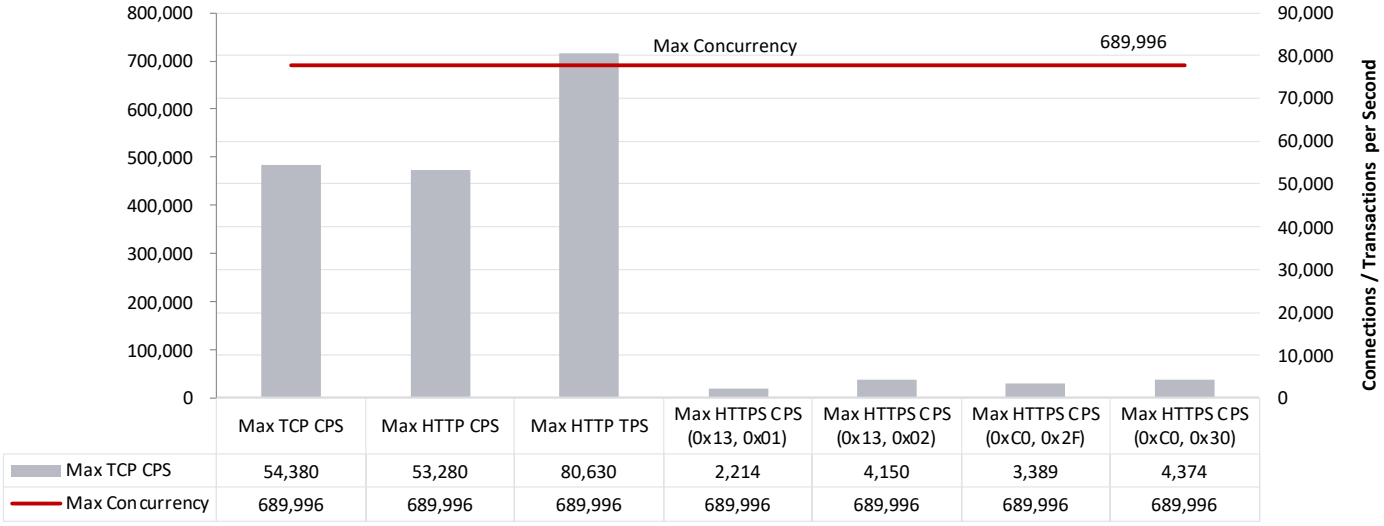


Figure 13 – Raw Packet Processing Performance (UDP Traffic)

MAXIMUM CAPACITY

These tests aimed to stress the inspection engine and determine how it copes with high volumes of TCP connections per second, application-layer transactions per second, and concurrent open connections. All packets contained valid payload and address data. Note that in all tests, final measurements were taken at the following critical "breaking points":

- Excessive concurrent TCP connections – Latency within the firewall is causing an increase in open connections.
- Excessive concurrent HTTP connections – Latency within the firewall is causing delays and increased response time.
- Unsuccessful HTTP transactions – Normally, there should be zero unsuccessful transactions. Once these appear, it indicates that firewall latency is causing connections to time out.

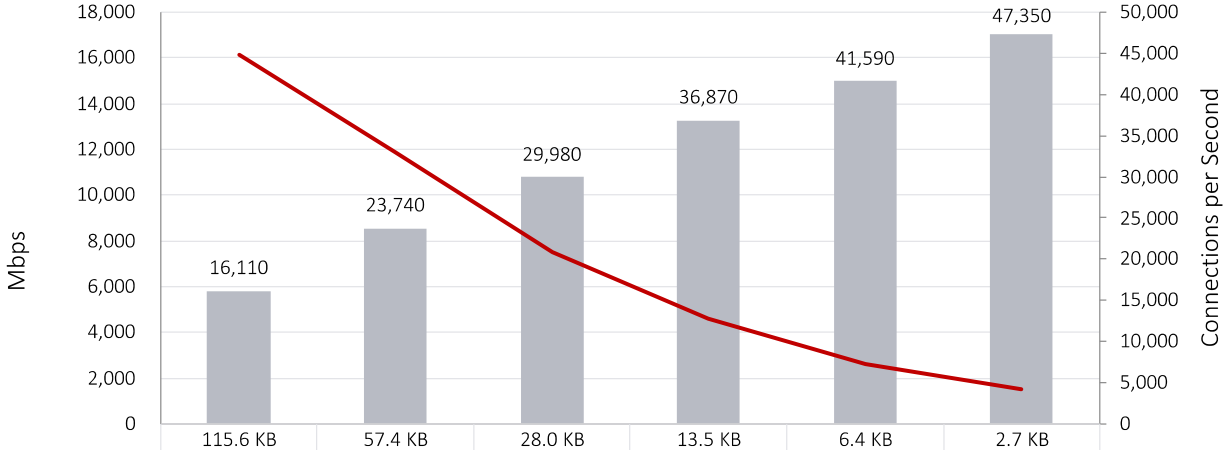


	Max TCP CPS	Max HTTP CPS	Max HTTP TPS	Max HTTPS CPS (0x13, 0x01)	Max HTTPS CPS (0x13, 0x02)	Max HTTPS CPS (0xC0, 0x2F)	Max HTTPS CPS (0xC0, 0x30)
Max TCP CPS	54,380	53,280	80,630	2,214	4,150	3,389	4,374
Max Concurrency	689,996	689,996	689,996	689,996	689,996	689,996	689,996

Figure 14 – Maximum Capacity

HTTP CAPACITY

The goal was to stress the HTTP detection engine and determine how the device copes with network loads of varying average packet size 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 rather than simple packet-based background traffic.



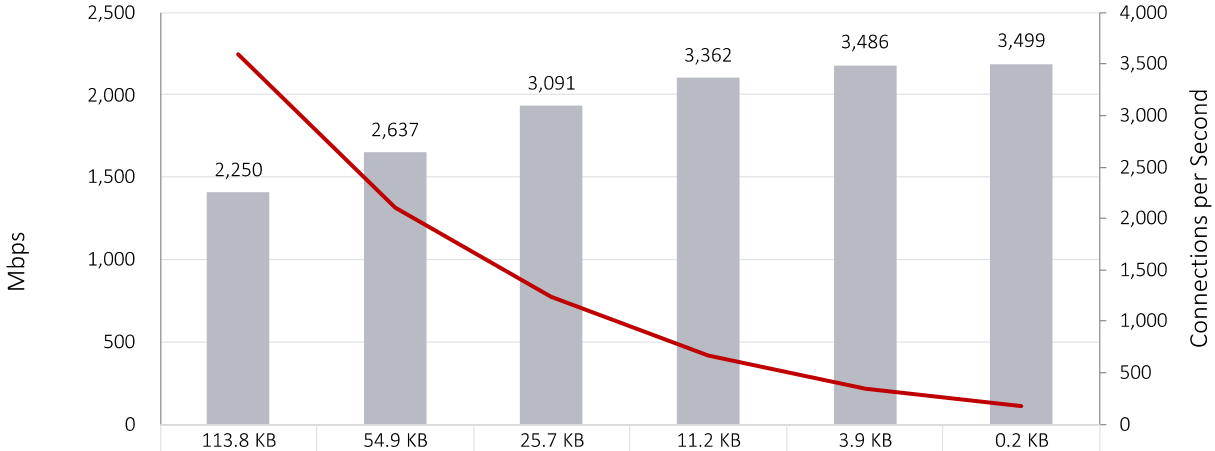
	115.6 KB	57.4 KB	28.0 KB	13.5 KB	6.4 KB	2.7 KB
Connections per Second (CPS)	16,110	23,740	29,980	36,870	41,590	47,350
Throughput (Mbps)	16,110	11,870	7,495	4,609	2,599	1,480
Average Response Time (ms)	0.6	0.6	0.6	0.6	0.6	0.6

Figure 15 – HTTP Capacity (Clear Text)

Each transaction consisted of a single HTTP GET request, and there were no transaction delays (i.e., the web server responded immediately to all requests). All packets contained valid payload (a mix of binary and ASCII objects) and address data. This test provided an excellent representation of a live network (albeit one biased towards HTTP traffic) at various network loads. For the application average response time, test traffic was passed across the infrastructure switches and through all inline port pairs of the device simultaneously (the basic infrastructure latency was known and constant throughout the tests).

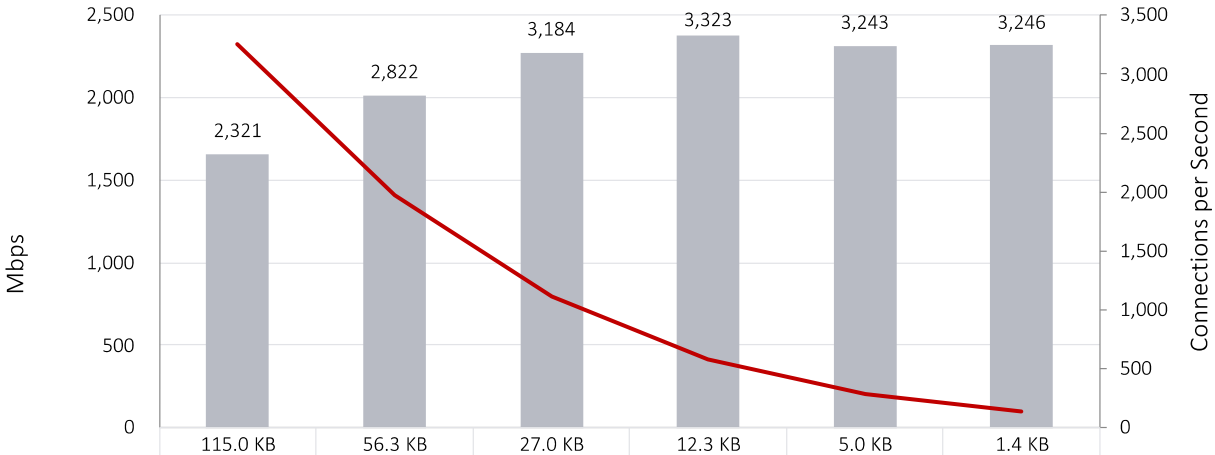
HTTPS CAPACITY

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 SSL/TLS with varying algorithms forced the device to decrypt traffic before inspection, increasing the workload further.



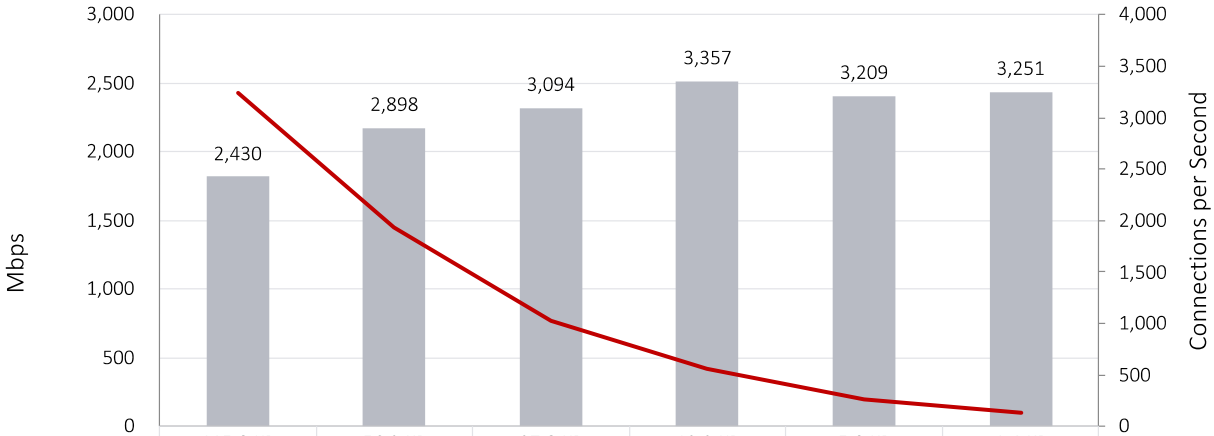
	113.8 KB	54.9 KB	25.7 KB	11.2 KB	3.9 KB	0.2 KB
Connections per Second (CPS)	2,250	2,637	3,091	3,362	3,486	3,499
Throughput (Mbps)	2,250	1,319	773	420	218	109
Average Response Time (ms)	24.9	13.8	12.3	7.5	4.2	3.4

Figure 16 – HTTPS Capacity [TLS_AES_256_GCM_SHA384 (0x13, 0x02)]



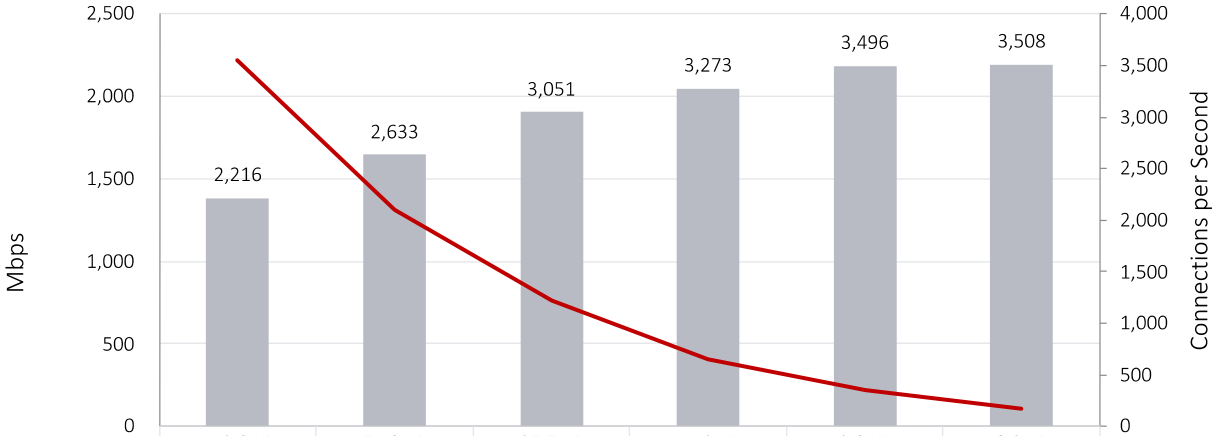
	115.0 KB	56.3 KB	27.0 KB	12.3 KB	5.0 KB	1.4 KB
Connections per Second (CPS)	2,321	2,822	3,184	3,323	3,243	3,246
Throughput (Mbps)	2,321	1,411	796	415	203	101
Average Response Time (ms)	353.2	284.9	159.6	84.2	16.1	15.6

Figure 17 – HTTPS Capacity [TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)]



	115.0 KB	56.3 KB	27.0 KB	12.3 KB	5.0 KB	1.4 KB
Connections per Second (CPS)	2,430	2,898	3,094	3,357	3,209	3,251
Throughput (Mbps)	2,430	1,449	774	420	201	102
Average Response Time (ms)	473.7	280.2	227.8	77.6	15.3	19.1

Figure 18 – HTTPS Capacity [TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)]



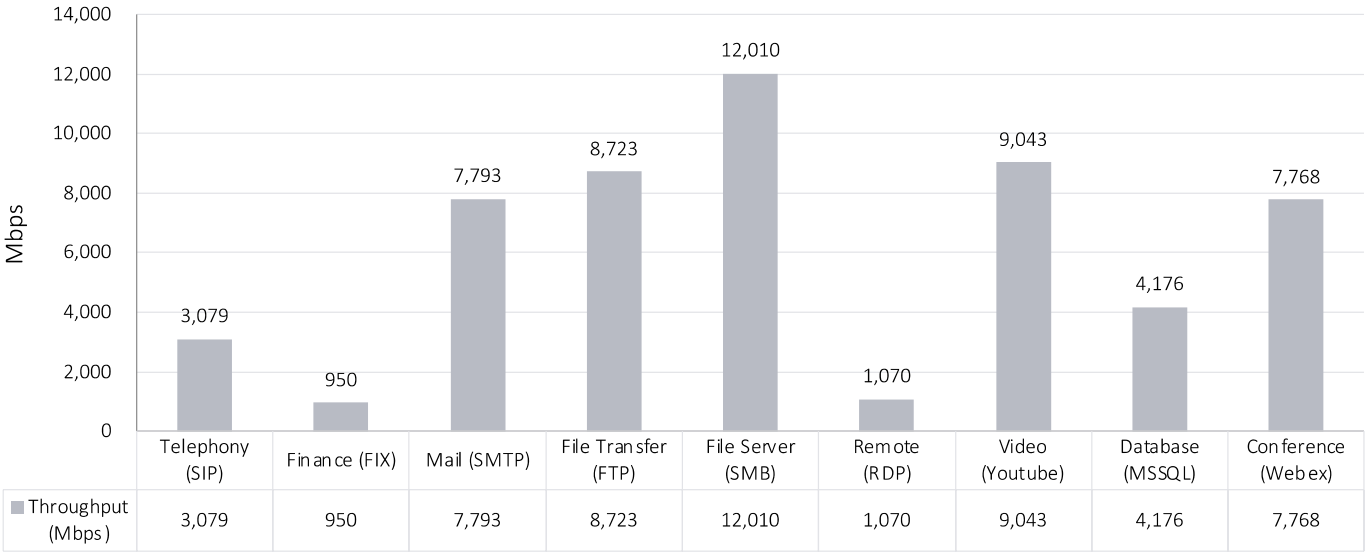
	113.8 KB	54.9 KB	25.7 KB	11.2 KB	3.9 KB	0.2 KB
Connections per Second (CPS)	2,216	2,633	3,051	3,273	3,496	3,508
Throughput (Mbps)	2,216	1,317	763	409	219	110
Average Response Time (ms)	20.4	12.7	9.6	5.7	3.4	3.3

Figure 19 – HTTPS Capacity [TLS_AES_128_GCM_SHA256 (0x13, 0x01)]

Tests were conducted with one transaction per connection; a single (1) HTTP(S) GET request. There were no transaction delays (the webserver responded immediately to all requests) and all packets contained valid payloads (a mix of binary and ASCII objects) and address data. Testing determined the maximum rate at which the firewall could process HTTPS packets 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 a load level of 95% of the maximum throughput with zero packet loss at each response size.

"REAL-WORLD" SINGLE APPLICATION FLOWS

Where previous tests provided a pure HTTP environment with varying connection rates and average packet sizes, this test aimed to simulate real-world single-application traffic.



	Telephony (SIP)	Finance (FIX)	Mail (SMTP)	File Transfer (FTP)	File Server (SMB)	Remote (RDP)	Video (Youtube)	Database (MSSQL)	Conference (Webex)
Throughput (Mbps)	3,079	950	7,793	8,723	12,010	1,070	9,043	4,176	7,768

Figure 20 – "Real-World" Single Application Flows

Stability & Reliability AAA

Long-term stability is essential for an inline device, where failure can produce network outages. These tests verified the firewall's stability and ability to maintain security effectiveness while under normal load and passing malicious traffic. A firewall that could not sustain legitimate traffic (or that crashed) while under hostile attack would 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 passes, caused by either the volume of traffic or by the product failing open for any reason, this results in a failure.

Stability & Reliability	Result
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
Protocol Fuzzing & Mutation	Pass

Figure 21 – Stability & Reliability

BLOCKING UNDER EXTENDED ATTACK

The network firewall was exposed to a constant stream of policy or protocol violations over an extended period. The product was configured to block and alert; thus, this test indicates the effectiveness of the flow management and alert handling mechanisms.

Blocking with Minimal Load

A continuous stream of security policy violations mixed with legitimate traffic was transmitted through the product for an extended period of time with no additional background traffic. This is not intended as a stress test for traffic load (covered in the performance section); it is a reliability test for consistency of blocking.

Blocking Under Load

This test provided an indication of the ability of the DUT to remain operational and stable (i.e., block violations and raise associated alerts) throughout a period of extended attack with load. This is intended as a stress test. This test adds legitimate traffic to the Blocking with Minimal Load test up to 90% of the maximums recorded in the HTTP, HTTPS, and Real-World Application Performance tests.

BEHAVIOR OF THE STATE ENGINE UNDER LOAD

This test determined whether the device was capable of preserving state across a large number of open connections over an extended time period. At various points throughout the test (including after the maximum has been reached), it was confirmed that the device was still capable of inspecting and blocking traffic that violated the currently applied security policy while confirming that legitimate traffic was not blocked (perhaps as a result of exhaustion of the resources allocated to state tables). The device must be able to apply policy decisions effectively based on inspected traffic at all load levels.

Attack Detection/Blocking – Normal Load

This test determined whether the device could detect and block policy violations as the number of open sessions reached 75% of the maximum determined in the Theoretical Maximum Concurrent TCP Connections performance test.

Pass Legitimate Traffic – Normal Load

This test ensured that the device continued to pass legitimate traffic as the number of open sessions reached 75% of the maximum determined in the Theoretical Maximum Concurrent TCP Connections performance test.

State Preservation – Normal Load

This test determined that the sensor maintained the state of pre-existing sessions as the number of open sessions reached 75% of the maximum determined in the Theoretical Maximum Concurrent TCP Connections performance test. A legitimate HTTP session was opened, and the first packet of a two-packet exploit was transmitted. As the number of open connections approached the maximum, the initial HTTP session was completed with the second half of the exploit, and the session was closed. If the firewall was still maintaining state of the original session, the exploit would be recorded and blocked. If the state tables have been exhausted and the connection was removed from the state table AND fails open (to a bypass condition), the exploit string will not be reconstructed properly and will not be detected as both halves of the exploit are required to trigger an alert. A product failed the test if it did not generate an alert after the second packet was transmitted or if it raised an alert on either half of the exploit on its own.

Drop Traffic – Maximum Exceeded

Did the firewall drop all excess traffic as the number of concurrent connections exceeded the maximum? This test ensured that the device continued to drop all traffic as the number of open sessions exceeded the maximum determined in the Theoretical Maximum Concurrent TCP Connections performance test.

Protocol Fuzzing & Mutation

This test stressed the protocol stacks of the firewall by exposing it to traffic from various protocol randomizer and mutation tools. Several of the tools in this category are based on the ISIC test suite. The device was expected to remain operational and capable of detecting and blocking exploits throughout the test.

Cost of Tested Configuration

Implementation of security solutions can be complex, with several factors affecting the overall cost of deployment, maintenance, and upkeep. The following should be considered over the course of the useful life of the solution:

- Product Purchase – The cost of acquisition
- Vendor Support – Fees paid to the vendor to provide support throughout the product life cycle
- Product Maintenance – The fees paid to the vendor, including software and hardware support, maintenance, and updates
- Implementation Time – Time required to install and configure the DUT in a production environment
- Upkeep – The time required to apply updates and patches from vendors, including hardware, software, and other updates
- Operational Management - Time commitment for day-to-day operations within a production environment, including support for monitoring logs, updating policies, and supporting incident investigations

PRICING OVER 3 YEARS

Calculations are based on public pricing information. The 24/7 maintenance and support option with 24-hour replacement is utilized wherever possible since enterprise customers typically select this option. Prices are for single-device management and maintenance only; central management solutions (CMS) costs may be extra.

Product	MSRP	24/7 Support	Total Cost (1-Year)	Total Cost (3-Years)
Forcepoint 2205 NGFW version 7.0.1.28052	$22,700	$3,405	$26,105	$32,915

Figure 22 – 3-Year Cost (US$)

- Year 1 Cost is calculated by adding installation costs + purchase price + first-year maintenance/support fees.
- Year 2 Cost consists only of maintenance/support fees.
- Year 3 Cost consists only of maintenance/support fees.

PRICE PER PROTECTED MBPS

One way to look at the value of a firewall is to think of it within the context of price/performance, or in this case, Price/Mbps.

$$\text{Price per Mbps} = \frac{\text{Price}}{\text{Mbps}}$$

Now that we have normalized the price within the context of performance, it is time to take into account that this is a security device. After all, an inexpensive device that only blocks 10 percent of attacks is not serving the purpose for which it was purchased. Therefore, calculating a security device's value requires considering the relationship between price, performance, and security; we take the Price/Mbps and divide it by Security Effectiveness. Using our formula, a device that provides less security, i.e., 50%, will be twice as expensive as a device with 100% security.

$$\text{Price per Protected Mbps} = \frac{\text{Price}}{\text{Security Effectiveness x Performance}}$$

(Security Effectiveness = Routing & access control x SSL/TLS Functionality x Threat Prevention x Stability & Reliability)

Product	3-Year Cost (Price)	Security Effectiveness	Rated Throughput	Price per Protected Mbps
Forcepoint 2205 NGFW version 7.0.1.28052	$32,915	99.48%	4,235 Mbps	$7.81

Figure 23 – Price per Protected Mbps

Appendix A – Scorecard

Summary			
Vendor		Forcepoint	
Device Model		2205	
Firmware		Forcepoint NGFW version 7.0.1.28052	
IPS Version		Update Package 1564	
Configuration		4 x 10G using 2 port pairs	
Rated Throughput (Max bandwidth 20 Gbps)		4,235	
Routing Functionality		**Result**	
Unrestricted Traffic Test		Pass	
Segmented Traffic Test		Pass	
Access Control		**Result**	
Simple Policies		Pass	
Complex Multi-Zone Policies		Pass	
SSL/TLS Support			
Cipher Suites	**Prevalence**	**Version**	**Result**
TLS_AES_256_GCM_SHA384 (0x13, 0x02)	63.90%	TLS 1.3	Pass
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)	13.70%	TLS 1.2	Pass
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)	9.90%	TLS 1.2	Pass
TLS_AES_128_GCM_SHA256 (0x13, 0x01)	7.70%	TLS 1.3	Pass
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA8)	1.30%	TLS 1.2	Pass
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28)	1.10%	TLS 1.2	Pass
TLS_CHACHA20_POLY1305_SHA256 (0x13, 0x03)	1.10%	TLS 1.3	Pass
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA9)	0.30%	TLS 1.2	Pass
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C)	0.30%	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			**Pass**
Decryption Bypass Exceptions			**Pass**
TLS Session Reuse - Session Tickets			**Pass**
TLS Session Reuse - Session IDs			**Pass**

Threat Prevention	
False Positives	**Result**
Browsing Test	100.00%
File Download Test	100.00%
Exploits	**Block Rate**
Exploits without Background Network Load	99.48%
Exploits with Background Network Load	99.48%
Evasions	**Result**
IP Address Spoofing	Pass
TCP Split Handshake Spoofing	Pass
IP Packet Fragmentation (Client & Server)	**Result**
(small IP fragments (32 bytes))	Pass
(small IP fragments (24 bytes))	Pass
(small IP fragments (16 bytes))	Pass
(small IP fragments (8 bytes))	Pass
(small IP fragments in order)	Pass
(small IP fragments in reverse order)	Pass
(small IP fragments in random order)	Pass
(small IP fragments; delay first fragment)	Pass
(small IP fragments; delay random fragment)	Pass
(small IP fragments; delay last fragment)	Pass
(overlapping small IP fragments)	Pass
(small IP fragments; interleave chaff (duplicate))	Pass
(small IP fragments; interleave chaff (duplicate with dummy payload) after fragment)	Pass
(small IP fragments; interleave chaff (invalid IP options) before fragment)	Pass
(small IP fragments; interleave chaff (invalid IP options) after fragment)	Pass
(small IP fragments; interleave chaff (invalid IP checksum) before fragment)	Pass
(small IP fragments; interleave chaff (invalid IP checksum) after fragment)	Pass
(small IP fragments (24-32 bytes))	Pass
(small IP fragments (16-24 bytes))	Pass
(small IP fragments (8-32 bytes))	Pass
(small IP fragments; delay first fragment; delay last fragment)	Pass
(small IP fragments in reverse order; delay first fragment; delay last fragment)	Pass
(small IP fragments; interleave chaff (invalid IP options) before and after fragment)	Pass
(small IP fragments; interleave chaff (invalid IP checksum) before and after fragment)	Pass
(overlapping small IP fragments favoring new data)	Pass
(small IP fragments; interleave chaff (duplicate with dummy payload) before fragment)	Pass
(overlapping small IP fragments favoring new data in reverse order)	Pass
(overlapping small IP fragments favoring new data in random order)	Pass
(overlapping small IP fragments favoring new data; interleave chaff (invalid IP options))	Pass
(overlapping small IP fragments favoring new data; interleave chaff (invalid IP checksum))	Pass
(overlapping small IP fragments favoring new data in reverse order; delay last fragment	Pass

(overlapping small IP fragments favoring new data in reverse order; delay last fragment; interleave chaff (invalid IP options))	Pass
(overlapping small IP fragments favoring new data in reverse order; delay last fragment; interleave chaff (invalid IP checksum))	Pass
(overlapping small IP fragments favoring new data in reverse order; delay last fragment; interleave chaff (invalid IP options); interleave chaff (invalid IP checksum))	Pass
(overlapping small IP fragments favoring new data in reverse order; delay last fragment; interleave chaff (invalid IP options); interleave chaff (invalid IP checksum); delay first fragment)	Pass
TCP Segmentation (Client & Server)	**Result**
(small TCP segments (1025 bytes))	Pass
(small TCP segments (1024 bytes))	Pass
(small TCP segments (1023 bytes))	Pass
(small TCP segments (513 bytes))	Pass
(small TCP segments (512 bytes))	Pass
(small TCP segments (511 bytes))	Pass
(small TCP segments (257 bytes))	Pass
(small TCP segments (256 bytes))	Pass
(small TCP segments (255 bytes))	Pass
(small TCP segments (129 bytes))	Pass
(small TCP segments (128 bytes))	Pass
(small TCP segments (127 bytes))	Pass
(small TCP segments (65 bytes))	Pass
(small TCP segments (64 bytes))	Pass
(small TCP segments (63 bytes))	Pass
(small TCP segments (33 bytes))	Pass
(small TCP segments (32 bytes))	Pass
(small TCP segments (31 bytes))	Pass
(small TCP segments (17 bytes))	Pass
(small TCP segments (16 bytes))	Pass
(small TCP segments (15 bytes))	Pass
(small TCP segments (9 bytes))	Pass
(small TCP segments (8 bytes))	Pass
(small TCP segments (7 bytes))	Pass
(small TCP segments (5 bytes))	Pass
(small TCP segments (4 bytes))	Pass
(small TCP segments (3 bytes))	Pass
(small TCP segments (2 bytes))	Pass
(small TCP segments (1 bytes))	Pass
(small TCP segments in order)	Pass
(small TCP segments in reverse order)	Pass
(small TCP segments in random order)	Pass
(small TCP segments (257 bytes); delay first segment)	Pass
(small TCP segments (256 bytes); delay first segment)	Pass
(small TCP segments (128 bytes); delay first segment)	Pass
(small TCP segments (64 bytes); delay first segment)	Pass

(small TCP segments; delay first segment)	Pass
(small TCP segments; delay random segment)	Pass
(small TCP segments; delay last segment)	Pass
(overlapping small TCP segments (16 bytes) favoring new data then non-overlapping small TCP segments (2 bytes))	Pass
(overlapping small TCP segments (16 bytes) favoring new data then overlapping small TCP segments (2 bytes) favoring old data)	Pass
(overlapping small TCP segments)	Pass
(overlapping small TCP segments favoring old data)	Pass
(overlapping small TCP segments favoring new data then non-overlapping small TCP segments)	Pass
(overlapping small TCP segments favoring new data then overlapping small TCP segments favoring old data)	Pass
(small TCP segments; interleave chaff (duplicate))	Pass
(small TCP segments; interleave chaff (invalid TCP checksums) before segment)	Pass
(small TCP segments; interleave chaff (invalid TCP checksums) after segment)	Pass
(small TCP segments; interleave chaff (older PAWS timestamps) after segment)	Pass
(small TCP segments; interleave chaff (out-of-window sequence numbers) before segment)	Pass
(small TCP segments; interleave chaff (out-of-window sequence numbers) after segment)	Pass
(small TCP segments; interleave chaff (requests to resynch sequence numbers mid-stream) before segment)	Pass
(small TCP segments; interleave chaff (requests to resynch sequence numbers mid-stream) after segment)	Pass
(small TCP segments (3-4 bytes))	Pass
(small TCP segments (2-3 bytes))	Pass
(small TCP segments (1-4 bytes))	Pass
(small TCP segments; delay first segment; delay last segment)	Pass
(small TCP segments in reverse order; delay first segment; delay last segment)	Pass
(small TCP segments; interleave chaff (invalid TCP checksums) before and after segment)	Pass
(small TCP segments; interleave chaff (out-of-window sequence numbers) before and after segment)	Pass
(small TCP segments; interleave chaff (requests to resynch sequence numbers mid-stream) before and after segment)	Pass
(overlapping small TCP segments favoring old data in random order)	Pass
(overlapping small TCP segments favoring old data; interleave chaff (invalid TCP checksums))	Pass
(overlapping small TCP segments favoring old data; interleave chaff (out-of-window sequence numbers))	Pass
(overlapping small TCP segments favoring old data; interleave chaff (requests to resynch sequence numbers mid-stream))	Pass
(overlapping small TCP segments favoring new data)	Pass
(small TCP segments; interleave chaff (older PAWS timestamps) before segment)	Pass
(overlapping small TCP segments favoring new data in random order)	Pass
(overlapping small TCP segments favoring new data; interleave chaff (invalid TCP checksums))	Pass
(overlapping small TCP segments favoring new data; interleave chaff (older PAWS timestamps))	Pass
(overlapping small TCP segments favoring new data; interleave chaff (out-of-window sequence numbers))	Pass
(overlapping small TCP segments favoring new data; interleave chaff (requests to resynch sequence numbers mid-stream))	Pass
Layered Evasions (Client & Server)	**Result**
(small TCP segments; small IP fragments)	Pass
(overlapping small TCP segments favoring old data; small IP fragments)	Pass
(overlapping small TCP segments favoring new data then non-overlapping small TCP segments ; small IP fragments)	Pass
(overlapping small TCP segments favoring new data then overlapping small TCP segments favoring old data ; small IP fragments)	Pass
(overlapping small TCP segments favoring new data; small IP fragments)	Pass

HTTP Obfuscation	Result
(HTTP/0.9 response (no response headers))	Pass
(Declared HTTP/0.9 response; but includes response headers; chunking declared but served without chunking)	Pass
(HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'))	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission (hex '04'))	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission block (hex '17'))	Pass
(HTTP/1.1 chunked response with chunk sizes followed by file separator (hex '1c'))	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a comma (hex '2c'))	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a $ (hex '24'))	Pass
(HTTP/1.1 chunked response with final chunk size of '000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000' (rather than '0'))	Pass
(HTTP/1.1 response with line folded transfer-encoding header declaring chunking ('Transfer-Encoding: ' followed by CRLF (hex '0d 0a') followed by 'chunked' followed by CRLF (hex '0d 0a'); served without chunking)	Pass
(HTTP/1.1 response with transfer-encoding header declaring chunking with lots of whitespace ('Transfer- Encoding:' followed by 8000 spaces (hex '20' * 8000) followed by 'chunked' followed by CRLF (hex '0d 0a'); served chunked)	Pass
(HTTP/1.0 response declaring chunking; served without chunking)	Pass
(HTTP/1.1 response with "Transfer-Encoding: chunked(hex 2C)"; served without chunking)	Pass
(HTTP/1.1 response with "Content-Encoding: gzip(hex 2C)"; served uncompressed)	Pass
(HTTP/1.0 response declaring chunking with invalid content-length header; served without chunking)	Pass
(HTTP/1.1 response with "\tTransfer-Encoding: chunked"; served chunked)	Pass
(HTTP/1.1 response with "\tTransfer-Encoding: chonked" after custom header line with "chunked" as value; served without chunking)	Pass
(HTTP/1.1 response with header with no field name and colon+junk string; followed by '\tTransfer-Encoding: chunked' header; followed by custom header; served chunked)	Pass
(HTTP/1.1 response with "\r\rTransfer-Encoding: chunked"; served chunked)	Pass
(HTTP/1.1\nTransfer-Encoding:chunked; header end \n\n; served chunked)	Pass
(HTTP/1.1 response with "SIP/2.0 200 OK\r\n" before status header; chunked)	Pass
(HTTP/1.1 response with space+junk string followed by \r\n before first header; chunked)	Pass
(HTTP/1.1 response with junk string before status header; chunked)	Pass
(HTTP/1.1 response with header end \n\004\n\n; chunked)	Pass
(HTTP/1.1 response with header end \r\n\010\r\n\r\n; chunked)	Pass
(HTTP/1.1 response with header end \n\r\r\n; chunked)	Pass
(HTTP/1.1 response with header end \n\006\011\n\n; chunked)	Pass
(HTTP/1.1 response with header end \n\033\n\003\n\n; chunked)	Pass
(HTTP/1.1 response with status code 202; with message-body; chunked)	Pass
(HTTP/1.1 response with status code 429; with message-body; chunked)	Pass
(HTTP/1.1 response with status code 300; with message-body; chunked)	Pass
(HTTP/1.1 response with status code 306; with message-body; chunked)	Pass
(HTTP/1.1 response with status code 414; with message-body; chunked)	Pass
(HTTP/1.1 chunked response with no status indicated)	Pass
(No status line; chunking indicated; served unchunked)	Pass
(HTTP/1.1 response with invalid content-length header size declaration followed by space and null (hex '20 00'))	Pass
(HTTP/2.0 declared; served chunked)	Pass
(HTTP/0001.1 declared; served chunked)	Pass

(HTTP/6.-66 declared; served chunked)	Pass
(HTTP/7.7 declared; served chunked)	Pass
(Double Transfer-Encoding: first empty; last chunked. Served with invalid content-length; not chunked.)	Pass
(Relevant headers padded by preceding with hundreds of random custom headers)	Pass
HTTP Compression	**Result**
(HTTP/1.1 response with content-encoding declaration of gzip followed by space+junk string; served uncompressed and chunked)	Pass
(HTTP/1.1 response with content-encoding header for deflate; followed by content-encoding header for gzip; served uncompressed and chunked)	Pass
(HTTP/1.1 response compressed with deflate)	Pass
(HTTP/1.1 response declaring deflate followed by junk string; served uncompressed)	Pass
(HTTP/1.1 response compressed with gzip)	Pass
(HTTP/1.1 response declaring gzip followed by junk string; served uncompressed)	Pass
(HTTP/1.1 response with "Transfer-Encoding: gzip"; served uncompressed)	Pass
(HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission (hex '04'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission block (hex '17'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by file separator (hex '1c'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a comma (hex '2c'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a $ (hex '24'); compressed with gzip)	Pass
(HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'); compressed with deflate)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission (hex '04'); compressed with deflate)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by end of transmission block (hex '17'); compressed with deflate)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by file separator (hex '1c'); compressed with deflate)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a comma (hex '2c'); compressed with deflate)	Pass
(HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a $ (hex '24'); compressed with deflate)	Pass
HTML Obfuscation	**Result**
(UTF-8 encoding)	Pass
(UTF-8 encoding with BOM)	Pass
(UTF-16 encoding with BOM)	Pass
(UTF-8 encoding; no http or html declarations)	Pass
(UTF-8 encoding with BOM; no http or html declarations)	Pass
(UTF-16 encoding with BOM; no http or html declarations)	Pass
(UTF-16-LE encoding)	Pass
(UTF-16-BE encoding)	Pass
(UTF-16-LE encoding; no http or html declarations)	Pass
(UTF-16-BE encoding; no http or html declarations)	Pass
(UTF-7 encoding)	Pass
(UTF-7 encoding; no http or html declarations)	Pass
(EICAR string included at top of HTML)	Pass
(padded with <5MB)	Pass
(padded with >5MB and <25MB)	Pass
(padded with >25MB)	Pass

(padded with >5MB and chunked)	Pass
(padded with >5MB and <25MB and chunked)	Pass
(padded with >25MB and chunked)	Pass
(padded with 5MB and compressed with gzip)	Pass
(padded with >5MB and <25MB and compressed with gzip)	Pass
(padded with >25MB and compressed with gzip)	Pass
(padded with 5MB and compressed with deflate)	Pass
(padded with >5MB and <25MB and compressed with deflate)	Pass
(padded with >25MB and compressed with deflate)	Pass
(UTF-8 encoding; padded with >25MB and chunked)	Pass
(UTF-8 encoding with BOM; padded with >25MB and chunked)	Pass
(UTF-16 encoding with BOM; padded with >25MB and chunked)	Pass
(UTF-8 encoding; no http or html declarations; padded with >25MB and chunked)	Pass
(UTF-8 encoding with BOM; no http or html declarations; padded with >25MB and chunked)	Pass
(UTF-16 encoding with BOM; no http or html declarations; padded with >25MB and chunked)	Pass
Combination	**Result**
(UTF-8 encoding; padded with >25MB and chunked; small TCP segments; small IP fragments)	Pass

Performance			
Raw Packet Processing Performance (UDP Throughput)		**Throughput (Mbps)**	**Latency (µs)**
64 Byte Frames		1,240	36.9
128 Byte Frames		2,040	39.8
256 Byte Frames		3,640	70.3
512 Byte Frames		5,640	63.7
1024 Byte Frames		8,040	75.6
1280 Byte Frames		8,440	82.7
1518 Byte Frames		9,240	93.3
Maximum Capacity	**CPS**	**TPS**	
Max Concurrent TCP Connection	689,996	-	
Max TCP CPS	54,380	-	
Max HTTP CPS	53,280	-	
Max HTTP TPS	-	80,630	
Max HTTPS CPS (0x13, 0x01)	2,214	-	
Max HTTPS CPS (0x13, 0x02)	4,150	-	
Max HTTPS CPS (0xC0, 0x2F)	3,389	-	
Max HTTPS CPS (0xC0, 0x30)	4,374	-	
HTTP Capacity	**CPS**	**Throughput (Mbps)**	**Response Time (ms)**
1,000 Connections Per Second - 115.6 KB Response	16,110	16,110	0.6
2,000 Connections Per Second - 57.4 KB Response	23,740	11,870	0.6
4,000 Connections Per Second - 28.0 KB Response	29,980	7,495	0.6
8,000 Connections Per Second - 13.5 KB Response	36,870	4,609	0.6
16,000 Connections Per Second - 6.4 KB Response	41,590	2,599	0.6
32,000 Connections Per Second - 2.7 KB Response	47,350	1,480	0.6
HTTPS Capacity (0x13, 0x02)	**CPS**	**Throughput (Mbps)**	**Response Time (ms)**
1,000 Connections Per Second - 113.8 KB Response	2,250	2,250	24.9
2,000 Connections Per Second - 54.9 KB Response	2,637	1,319	13.8
4,000 Connections Per Second - 25.7 KB Response	3,091	773	12.3
8,000 Connections Per Second - 11.2 KB Response	3,362	420	7.5
16,000 Connections Per Second - 3.9 KB Response	3,486	218	4.2
32,000 Connections Per Second - 0.2 KB Response	3,499	109	3.4
HTTPS Capacity (0xC0, 0x30)	**CPS**	**Throughput (Mbps)**	**Response Time (ms)**
1,000 Connections Per Second - 115.0 KB Response	2,321	2,321	353.2
2,000 Connections Per Second - 56.3 KB Response	2,822	1,411	284.9
4,000 Connections Per Second - 27.0 KB Response	3,184	796	159.6
8,000 Connections Per Second - 12.3 KB Response	3,323	415	84.2
16,000 Connections Per Second - 5.0 KB Response	3,243	203	16.1
32,000 Connections Per Second - 1.4 KB Response	3,246	101	15.6

HTTPS Capacity (0xC0, 0x2F)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 115.0 KB Response	2,430	2,430	473.7
2,000 Connections Per Second - 56.3 KB Response	2,898	1,449	280.2
4,000 Connections Per Second - 27.0 KB Response	3,094	774	227.8
8,000 Connections Per Second - 12.3 KB Response	3,357	420	77.6
16,000 Connections Per Second - 5.0 KB Response	3,209	201	15.3
32,000 Connections Per Second - 1.4 KB Response	3,251	102	19.1
HTTPS Capacity (0x13, 0x01)	**CPS**	**Throughput (Mbps)**	**Response Time (ms)**
1,000 Connections Per Second - 113.8 KB Response	2,216	2,216	20.4
2,000 Connections Per Second - 54.9 KB Response	2,633	1,317	12.7
4,000 Connections Per Second - 25.7 KB Response	3,051	763	9.6
8,000 Connections Per Second - 11.2 KB Response	3,273	409	5.7
16,000 Connections Per Second - 3.9 KB Response	3,496	219	3.4
32,000 Connections Per Second - 0.2 KB Response	3,508	110	3.3

Real-World Single Application Flows	Throughput (Mbps)
Telephony (SIP)	3,079
Finance (FIX)	950
Mail (SMTP)	7,793
File Transfer (FTP)	8,723
File Server (SMB)	12,010
Remote (RDP)	1,070
Video (Youtube)	9,043
Database (MSSQL)	4,176
Web Conference (Webex)	7,768

Stability and Reliability	Result
Drop Traffic – Maximum Exceeded	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
Protocol Fuzzing & Mutation	Pass

Appendix C – CyberRatings Classification Matrix

Rating	Definition
AAA	A product rated 'AAA' has the highest rating assigned by CyberRatings. The product's capacity to meet its commitments to consumers is extremely strong.
AA	A product rated 'AA' differs from the highest-rated products only to a small degree. The product's capacity to meet its commitments to consumers is very strong.
A	A product rated 'A' is somewhat less capable than higher-rated categories. However, the product's capacity to meet its commitments to consumers is still strong
BBB	A product rated 'BBB' exhibits adequate stability and reliability. However, previously unseen events and use cases are more likely to negatively impact the product's capacity to meet its commitments to consumers.
	A product rated 'BB,' 'B,' 'CCC,' 'CC,' and 'C' is regarded as having significant risk characteristics. 'BB' indicates the least degree of risk and 'C' the highest. While such products will likely have some specialized capability and features, these may be outweighed by large uncertainties or major exposure to adverse conditions.
BB	A product rated 'BB' is more susceptible to failures than products that have received higher ratings. The product has the capacity to meet its commitments to consumers. However, it faces minor technical limitations that have a potential to be exposed to risks.
B	A product rated 'B' is more susceptible to failures than products rated 'BB'; however, it has the minimum capacity. Adverse conditions will likely expose the product's technical limitations that lead to an inability to meet its commitments to consumers.
CCC	A product rated 'CCC' is susceptible to failures and is dependent upon favorable conditions to perform expected functions. In the event of adverse conditions, the product is not likely to have the capacity to meet its commitments to consumers.
CC	A product rated 'CC' is highly susceptible to failures. The 'CC' rating is used when a failure has not yet occurred, but CyberRatings considers it a virtual certainty.
C	A product rated 'C' is highly susceptible to failures. The product is expected to fail under any abnormal operating conditions and does not offer a useful management systems and logging information compared with products that are rated higher.
D	A product rated 'D' is actively underperforming and failing and does not meet the use-case. The 'D' rating is used when the product is not operational without a major technical overhaul. Unless CyberRatings believes that such technical fixes will be made within a stated grace period (typically 30-90 calendar days), the 'D' rating also is an indicator that existing customers using the product have already experienced a failure and should take immediate action

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 Enterprise Firewall.

We would also like to thank TeraPackets for providing us with their Threat Replayer tool.

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