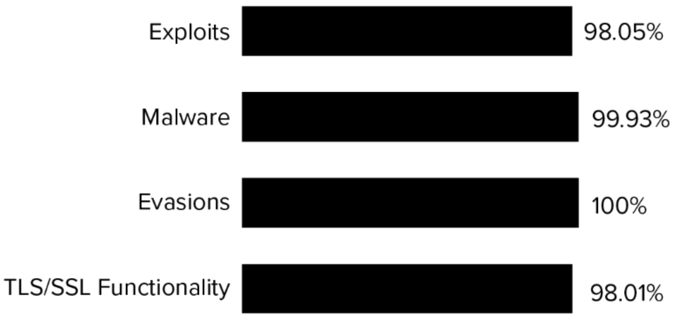
Zscaler Zero Trust Exchange

AAA

OVERVIEW

In Q2 2024, CyberRatings.org performed an independent test of Zscaler Zero Trust Exchange against the Security Service Edge (SSE) Threat Protection Methodology v2.1 using Amazon Web Services and 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 205 exploits, 7,140 malware samples, and whether any of 1,124 evasions could bypass its protection. Both clear text and encrypted traffic were measured to provide a more realistic rating based on modern network traffic.

98.0% PROTECTION RATE



THREAT PREVENTION

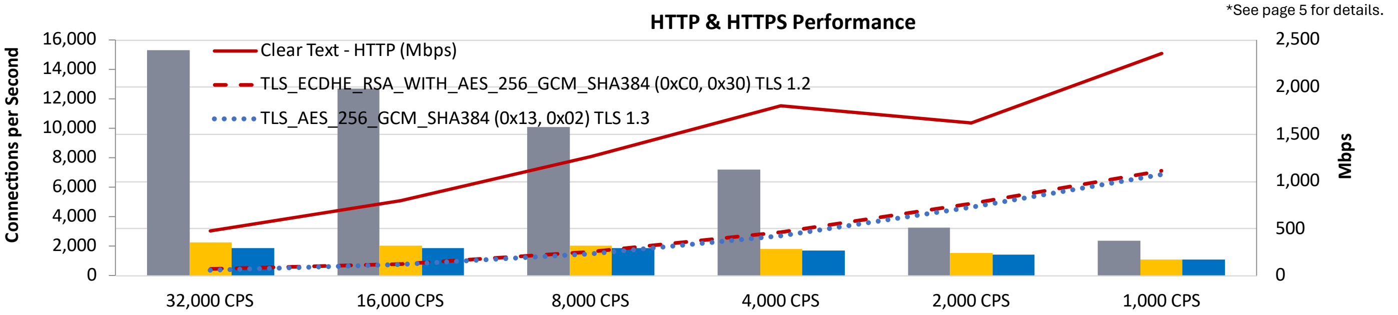
Threats:	Blocked	Tested
Exploits	201	205
Malware	7135	7140
Wild Malware – w/o Reputation	6191	6195
Wild Malware – w/ Reputation	944	945
Evasions	1124	1124
HTTP	602	602
HTML	108	108
Malware Evasions	290	290
Java	64	64
Combination	60	60

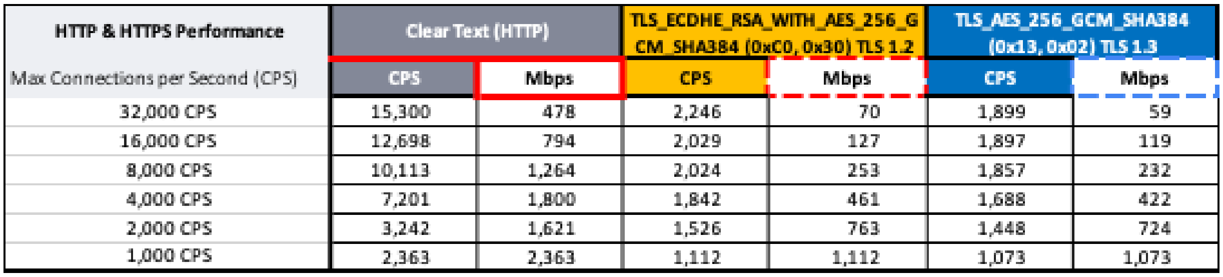
TLS/SSL DECRYPTION FUNCTIONALITY

Version	Prevalence	Cipher Suites	Results
TLS 1.3	66.51%	(0x13, 0x02)	Supported
TLS 1.2	11.85%	(0xC0, 0x30)	Supported
TLS 1.2	9.26%	(0xC0, 0x2F)	Supported
TLS 1.3	8.07%	(0x13, 0x01)	Supported
TLS 1.2	1.72%	(0xCC, 0xA8)	Not Supported
TLS 1.2	0.68%	(0xC0, 0x28)	Supported
TLS 1.3	0.55%	(0x13, 0x03)	Supported
TLS 1.2	0.42%	(0xC0, 0x2C)	Supported*
TLS 1.2	0.27%	(0xCC, 0xA9)	Not Supported
TLS 1.2	0.20%	(0xC0, 0x2B)	Supported*

*See page 5 for details.

THROUGHPUT





HTTP & HTTPS Performance	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	
Max Connections per Second (CPS)	CPS	Mbps	CPS	Mbps	CPS	Mbps
32,000 CPS	15,300	478	2,246	70	1,899	59
16,000 CPS	12,698	794	2,029	127	1,897	119
8,000 CPS	10,113	1,264	2,024	253	1,857	232
4,000 CPS	7,201	1,800	1,842	461	1,688	422
2,000 CPS	3,242	1,621	1,526	763	1,448	724
1,000 CPS	2,363	2,363	1,112	1,112	1,073	1,073

Threat Prevention

An SSE is a purpose-built cloud platform of integrated network security services designed to facilitate 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.

False Positives

Browsing 99.86% (1,417/1,419)

File Download 96.85% (1,690/1,745)

A key to effective protection is correctly identifying and allowing legitimate traffic while protecting against malware, exploits, and phishing attacks. False positives are any legitimate, non-malicious content/traffic perceived as malicious. False positive tests assessed the Zscaler Zero Trust Exchange's ability to block attacks while permitting legitimate traffic. If the SSE experienced false positive events, it was tuned until no further false positive events were encountered.

Exploit Protection

98.05% Blocked (201/205)

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

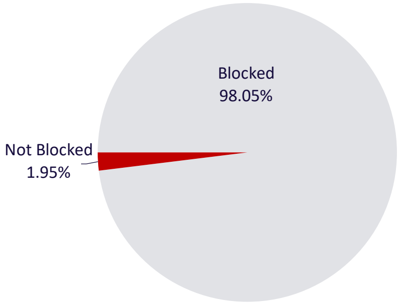


Figure 2 – Exploit Block Rate

Coverage by Target Vendor

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

Vendor	Coverage %
Adobe	94.12%
Apache	100%
Cisco	100%
Foxit	100%
Google	100%
LibreOffice	100%
Microsoft	100%
OMRON	100%
Oracle	100%
VMware	100%

Figure 1 — Coverage by Target Vendor

Coverage by Date

Coverage by date 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 six years.

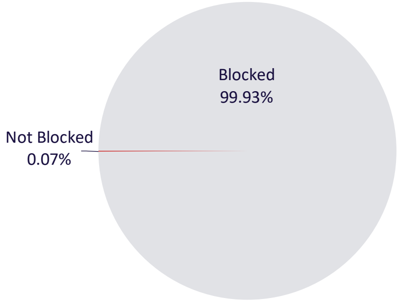
Year	Coverage %
2018	100%
2019	98.52%
2020	100%
2021	91.67%
2022	88.89%
2023	100%

Figure 3 — Coverage by Date

Malware Protection

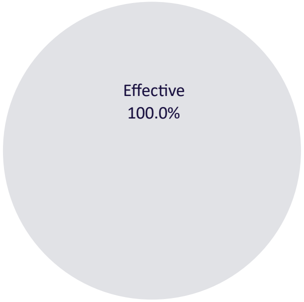
99.93% Blocked (7,135/7,140)

CyberRatings defines malware as software designed to disrupt, damage, or gain unauthorized access to computer systems. Malware can take many forms, including viruses, worms, Trojan horses, ransomware, spyware, adware, and other malicious programs. Its primary goal is to compromise the confidentiality, integrity, or availability of the victim's data or system.



Resistance to Evasions

100% Effective (1,124/1,124)



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

Handling evasions is hard. Our engineers verified that the Zscaler Zero Trust Exchange could block exploits and malware 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, HTTP evasions can be more broadly applied than HTML evasions. An HTTP evasion can be applied to thousands of exploits whereas a Java evasion is limited to fewer exploits.

We used multiple exploits and malware samples for each evasion technique during testing to see how the SSE defended against these combinations. Exploits and malware were tested across HTTP and HTTPS to see if the SSE could correctly decrypt and inspect each attack.

Evasion Technique	Number of Evasions Tested	Number of Evasions Blocked
HTTP	602	602
HTML	108	108
Malware Evasions (Packers, compressors, and portable executable)	290	290
Java	64	64
Combination	60	60

Figure 6 – Evasions by Technique

TLS/SSL Functionality

The use of the Secure Sockets Layer (SSL) protocol and its current iteration, Transport Layer Security (TLS), are 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 SSE were 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)	Supported
TLS 1.2	11.85%	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)	Supported
TLS 1.2	9.26%	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)	Supported
TLS 1.3	8.07%	TLS_AES_128_GCM_SHA256 (0x13, 0x01)	Supported
TLS 1.2	1.72%	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA8)	Not Supported
TLS 1.2	0.68%	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28)	Supported
TLS 1.3	0.55%	TLS_CHACHA20_POLY1305_SHA256 (0x13, 0x03)	Supported
TLS 1.2	0.42%	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C)	Supported*
TLS 1.2	0.27%	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA9)	Not Supported
TLS 1.2	0.20%	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B)	Supported*

Figure 7 – TLS/SSL Functionality

First, we tested to verify the SSE correctly inspected and blocked prohibited content. We then encrypted using the top 10 most prevalent ciphers and verified that the prohibited content was still inspected and blocked. If a cipher suite was not supported, we verified the SSE blocked all traffic using that cipher. Otherwise, an attacker could simply bypass security using an unsupported cipher suite.

* The ECDSA (Elliptic Curve Digital Signature Algorithm) authentication algorithm is supported only on the server-side (Server to Service Edge) encrypted connections.[3]

[1] Let's Encrypt Stats (https://letsencrypt.org/stats/)

[2] https://crawler.ninja/files/ciphers.txt

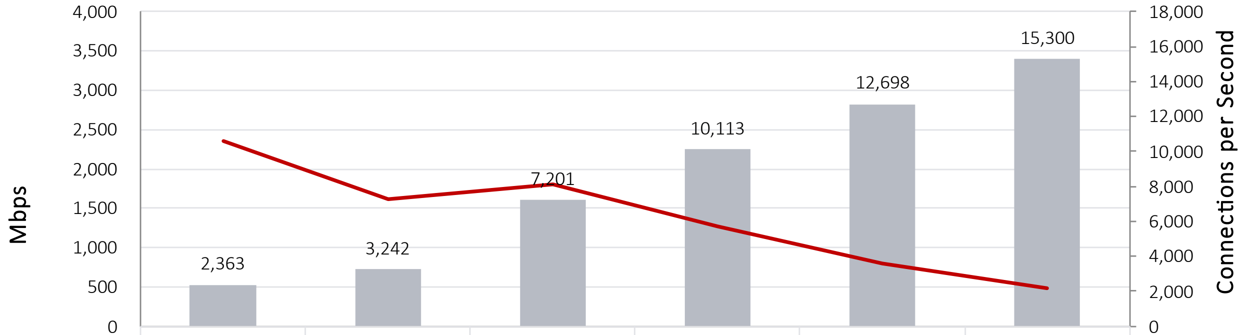
[3] https://help.zscaler.com/zia/supported-cipher-suites-ssl-inspection

Performance

Cloud security architects are tasked with designing environments that scale. The performance of the SSE 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 SSE is appropriate for a given environment. The performance tests were conducted at various locations across the United States. The results may differ depending on factors such as the geographical distance between clients and servers, the tunneling protocols employed, the bandwidth of the tunnels, the internet connectivity between sites, and the server's capacity to handle high CPS (connections per second) and throughput.

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.



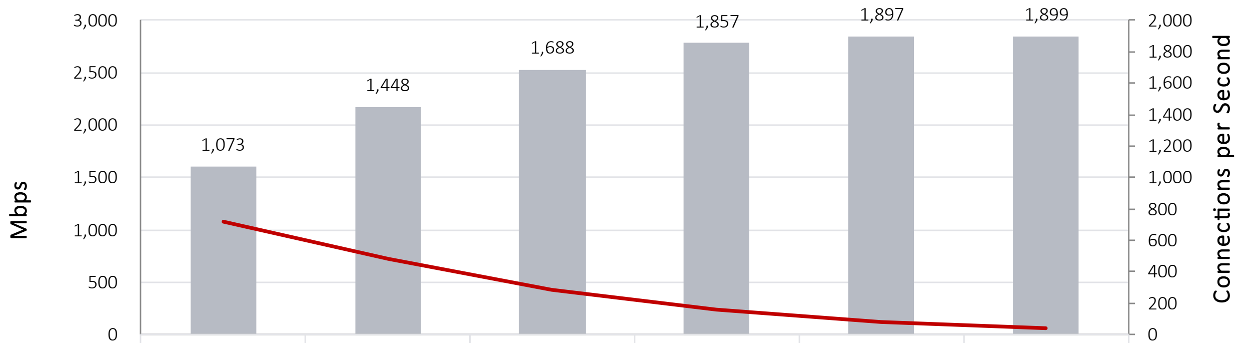
	115.6 KB	57.4 KB	28.0 KB	13.5 KB	6.4 KB	2.7 KB
▬ Connections per Second (CPS)	2,363	3,242	7,201	10,113	12,698	15,300
━ Throughput (Mbps)	2,363	1,621	1,800	1,264	794	478
Response Time (ms)	3,633.1	1,337.9	924.9	383.7	280.6	174.1

Figure 8 – 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.

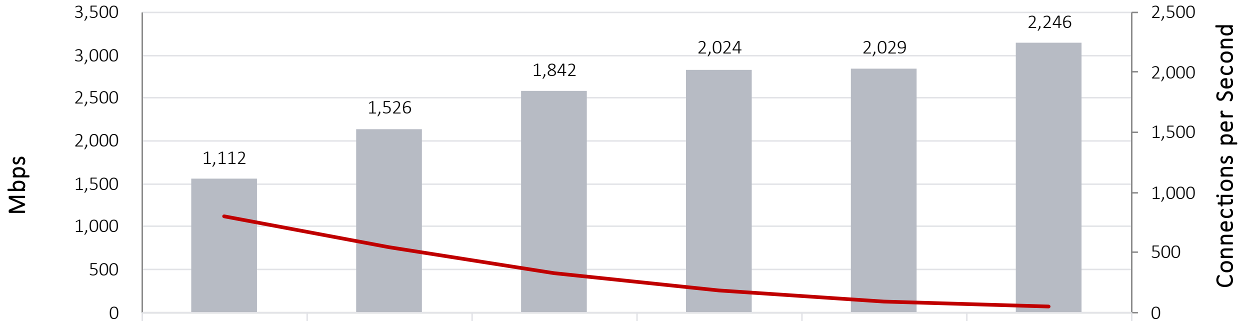
HTTPS Capacity

The goal was to stress the HTTPS engine and determine how the SSE coped with network loads of varied packet sizes and varying connections per second. The SSE was forced to track valid TCP sessions by creating session-based traffic with varying session lengths, 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.



	113.8 KB	54.9 KB	25.7 KB	11.2 KB	3.9 KB	0.2 KB
▨ Connections per Second (CPS)	1,073	1,448	1,688	1,857	1,897	1,899
— Throughput (Mbps)	1,073	724	422	232	119	59
Response Time (ms)	2,671.6	1,547.7	843.6	0.0	0.0	0.0

Figure 9 – 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)	1,112	1,526	1,842	2,024	2,029	2,246
— Throughput (Mbps)	1,112	763	461	253	127	70
Response Time (ms)	3,218.2	1,953.0	1,006.7	0.0	0.0	0.0

Figure 10 – HTTPS Capacity [TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)]

Download Test

As many security devices and services can impact the time it takes to download files (e.g., PDFs, data files, zipped files, documents, etc.), it is important to understand the impact on files in a variety of formats as they are downloaded.

Files from each of the following types were downloaded from the following locations to a local folder:

- Microsoft Office Word files
- Microsoft Office Excel files
- Adobe Acrobat PDFs
- WinZip Zipped files/folders

This test was first performed without the SSE to establish a baseline. SSE was then enabled, and the test was rerun. Thus, the results are relative to the baseline. The net increase in time to copy clean files of various sizes is determined.

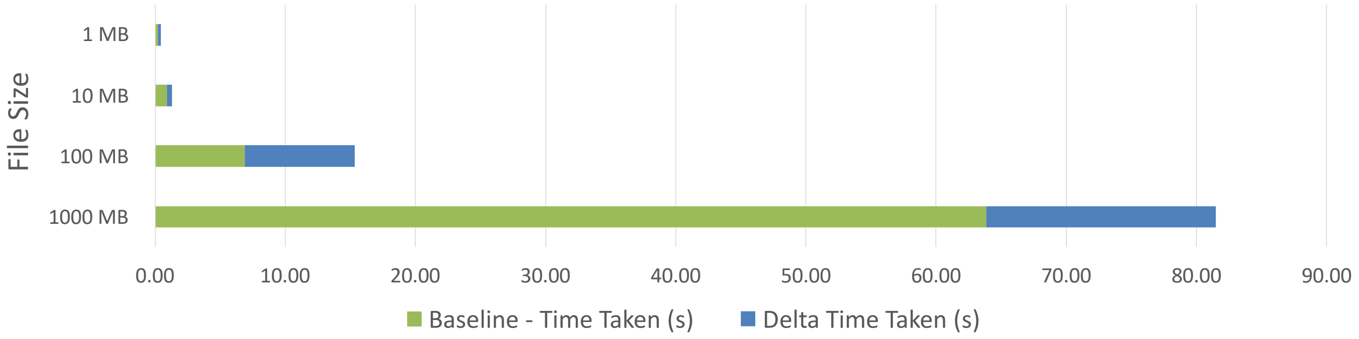


Figure 11 – Average Download of files from baseline and SSE

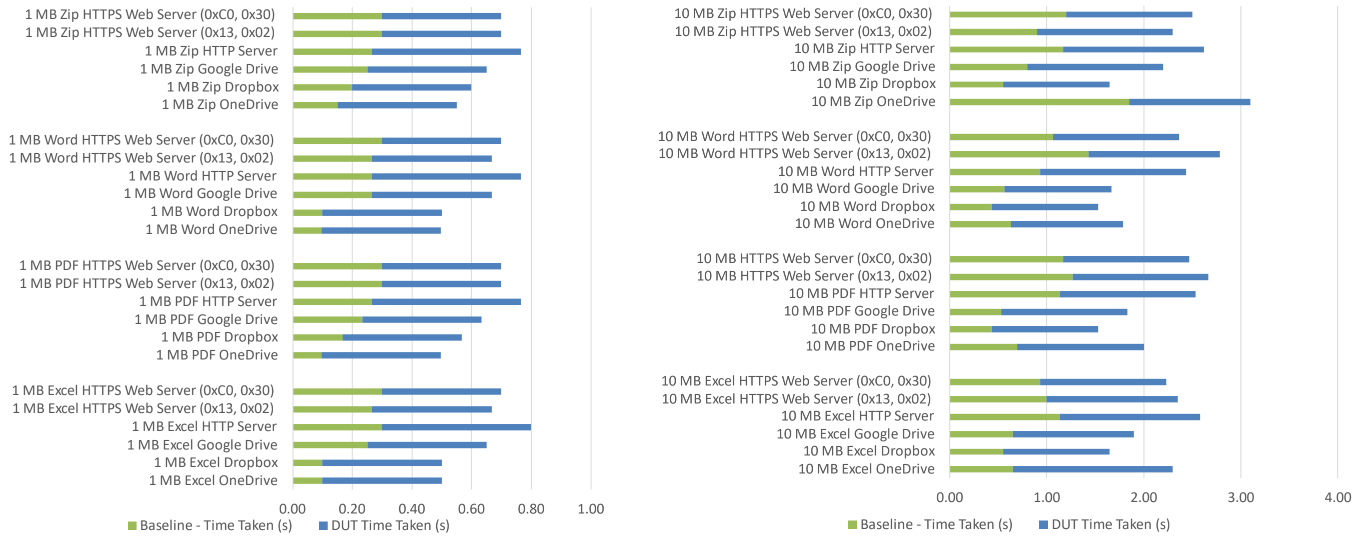


Figure 12 – Download of 10 Mb and 100 Mb files from baseline and SSE

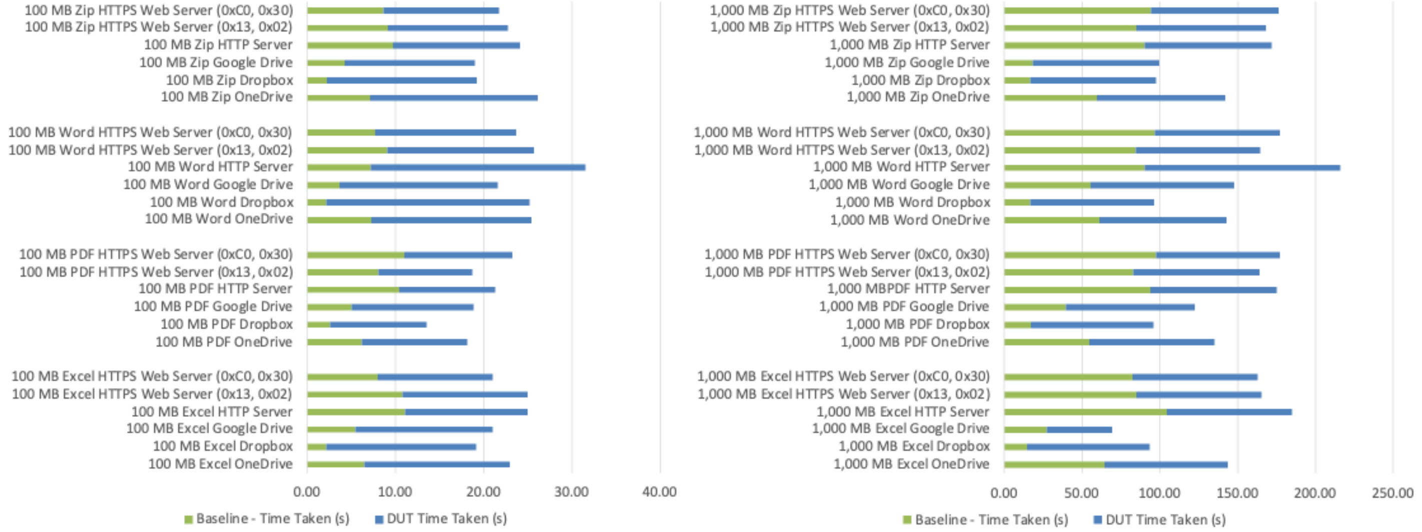


Figure 13 – Download of 100 Mb and 1,000 Mb files from baseline and SSE

Scorecard

TLS/SSL Support			
Cipher Suites	Prevalence	Version	Result
TLS_AES_256_GCM_SHA384 (0x13, 0x02)	66.51%	TLS 1.3	Supported
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)	11.85%	TLS 1.2	Supported
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)	9.26%	TLS 1.2	Supported
TLS_AES_128_GCM_SHA256 (0x13, 0x01)	8.07%	TLS 1.3	Supported
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA8)	1.72%	TLS 1.2	Not Supported
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28)	0.68%	TLS 1.2	Supported
TLS_CHACHA20_POLY1305_SHA256 (0x13, 0x03)	0.55%	TLS 1.3	Supported
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C)	0.42%	TLS 1.2	Supported
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA9)	0.27%	TLS 1.2	Not Supported
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B)	0.20%	TLS 1.2	Supported

Threat Prevention	
False Positives	Result
File Download Test	96.85%
Browsing Test	99.86%
Exploits	Block Rate
Exploits without Background Network Load	98.05%
Exploits with Background Network Load	98.05%
Malware	Block Rate
Wild Malware – w/o Reputation	99.94%
Wild Malware – w/ Reputation	99.89%
Evasions	Result
All Evasions	100%
HTTP	100%
HTML	100%
Malware Evasions	100%
Java	100%
Combination	100%
Evasion Detail	Result
7z with high compression using the BZIP2 algorithm (CL=9)	Pass
7z with high compression using the PPMD algorithm (CL=9)	Pass

Add HTTP header (field=HTTP/1.0) (value=HTTP/1.0) (before)	Pass
Add HTTP header (field=X-Content-Encoding) (value=gzip) (after)	Pass
Add HTTP header (field=X-Forwarded-For) (value=127.0.0.1) (after)	Pass
Add HTTP header (field=X-Padding) (after)	Pass
Add HTTP header (field=X-Test) (value=X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*) (before)	Pass
Add HTTP header (field=X-Transfer-Encoding) (value=chunked) (after)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add HTTP header (field=X-Padding) (after)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:A)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:A); Add HTTP header (field=X-Padding) (after)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:A); HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (before)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:A); HTTP Deflate Compression Content Encoding	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:A); HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte); Prefix the status line with (0x0d0a0d0a)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:random)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:random); Add HTTP header (field=X-Padding) (after)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:random); HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (before)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:random); HTTP Deflate Compression Content Encoding	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:random); HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte); Prefix the status line with (0x0d0a0d0a)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:A)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:A); Add HTTP header (field=X-Padding) (after)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:A); HTTP Brotli Compression Content Encoding	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:A); HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:A); HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (before)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:random)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:random); Add HTTP header (field=X-Padding) (after)	Pass

Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:random); HTTP Brotli Compression Content Encoding	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:random); HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:random); HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (before)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); HTTP Brotli Compression Content Encoding	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (before)	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); HTTP Deflate Compression Content Encoding	Pass
Add newline padding to each newline in JavaScript (size: 100 newlines); HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte); Prefix the status line with (0x0d0a0d0a)	Pass
Add newline padding to each newline in JavaScript (size: 1000 newlines)	Pass
Add padding to the document (size: 10000 bytes) (position: %{position}) (padding bytes:A)	Pass
Add padding to the document (size: 10000 bytes) (position: %{position}) (padding bytes:random)	Pass
Add padding to the document (size: 10000 bytes) (position: before) (padding bytes:A)	Pass
Add padding to the document (size: 10000 bytes) (position: before) (padding bytes:random)	Pass
Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:A)	Pass
Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:A); Add HTTP header (field=X-Padding) (after)	Pass
Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:A); HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (before)	Pass
Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:A); HTTP Deflate Compression Content Encoding	Pass
Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:A); HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte); Prefix the status line with (0x0d0a0d0a)	Pass
Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:random)	Pass
Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:random); Add HTTP header (field=X-Padding) (after)	Pass
Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:random); HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (before)	Pass
Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:random); HTTP Deflate Compression Content Encoding	Pass
Add padding to the document (size: 20000 bytes) (position: %{position}) (padding bytes:random); HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte); Prefix the status line with (0x0d0a0d0a)	Pass
Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:A)	Pass
Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:A); Add HTTP header (field=X-Padding) (after)	Pass
Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:A); HTTP Brotli Compression Content Encoding	Pass

Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:A); HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:A); HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (before)	Pass
Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:random)	Pass
Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:random); Add HTTP header (field=X-Padding) (after)	Pass
Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:random); HTTP Brotli Compression Content Encoding	Pass
Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:random); HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
Add padding to the document (size: 20000 bytes) (position: before) (padding bytes:random); HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (before)	Pass
Bz2 with high compression (CL=9)	Pass
Declared HTTP/0.9 response; but includes response headers; chunking declared but served without chunking	Pass
Double Transfer-Encoding: first empty; last chunked. Served with invalid content-length; not chunked.	Pass
EICAR string included at top of HTML	Pass
Gz with high compression (CL=9)	Pass
HTTP Brotli Compression Content Encoding	Pass
HTTP Brotli Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
HTTP Chunked Transfer Encoding (1-byte)	Pass
HTTP Chunked Transfer Encoding (1024-byte)	Pass
HTTP Chunked Transfer Encoding (16-byte)	Pass
HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (!!!!) (after)	Pass
HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (!!!!1) (after)	Pass
HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (.9) (after)	Pass
HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (.999999999999999999999) (after)	Pass
HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (0) (before)	Pass
HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (0000000000000000) (before)	Pass
HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (before)	Pass
HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (0!)	Pass
HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (0+0)	Pass
HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (000000000)	Pass
HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header	Pass
HTTP Chunked Transfer Encoding (2-byte)	Pass
HTTP Chunked Transfer Encoding (256-byte)	Pass

HTTP Chunked Transfer Encoding (3-byte)	Pass
HTTP Chunked Transfer Encoding (32-byte)	Pass
HTTP Chunked Transfer Encoding (4-byte)	Pass
HTTP Chunked Transfer Encoding (5-byte)	Pass
HTTP Chunked Transfer Encoding (5-byte); Prefix the status line with (0x0d0a)	Pass
HTTP Chunked Transfer Encoding (5-byte); Prefix the status line with (0x0d0a0d0a)	Pass
HTTP Chunked Transfer Encoding (512-byte)	Pass
HTTP Chunked Transfer Encoding (64-byte)	Pass
HTTP Chunked Transfer Encoding (8-byte)	Pass
HTTP Deflate Compression Content Encoding	Pass
HTTP Deflate Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
HTTP Gzip Compression Content Encoding	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (32-byte); Add HTTP header (field=Transfer-Encoding) (value=identity) (after)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte); Prefix the status line with (0x0d0a0d0a)	Pass
HTTP Identity Content Encoding	Pass
HTTP Identity Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
HTTP Identity Transfer Encoding	Pass
HTTP/0.9 response (no response headers)	Pass
HTTP/0001.1 declared; served chunked	Pass
HTTP/1.0 response declaring chunking with invalid content-length header; served without chunking	Pass
HTTP/1.0 response declaring chunking; served without chunking	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 comma (hex '2c'); compressed with deflate	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')	Pass
HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a $ (hex '24'); compressed with deflate	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 followed by end of transmission (hex '04')	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 (hex '04'); compressed with gzip	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 end of transmission block (hex '17'); compressed with deflate	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')	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 file separator (hex '1c'); compressed with gzip	Pass
HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30')	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 preceded by multiple zeros (hex '30'); compressed with gzip	Pass
HTTP/1.1 chunked response with final chunk size of '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000' (rather than '0')	Pass
HTTP/1.1 chunked response with no status indicated	Pass
HTTP/1.1 response compressed with deflate	Pass
HTTP/1.1 response compressed with gzip	Pass
HTTP/1.1 response declaring deflate followed by junk string; served uncompressed	Pass
HTTP/1.1 response declaring gzip followed by junk string; served uncompressed	Pass
HTTP/1.1 response with "\r\rTransfer-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 "\tTransfer-Encoding: chunked"; served chunked	Pass
HTTP/1.1 response with "Content-Encoding: gzip(hex 2C)"; served uncompressed	Pass
HTTP/1.1 response with "SIP/2.0 200 OK\r\n" before status header; chunked	Pass
HTTP/1.1 response with "Transfer-Encoding: chunked(hex 2C)"; served without chunking	Pass
HTTP/1.1 response with "Transfer-Encoding: gzip"; served uncompressed	Pass
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 with header end \n\004\n\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 header end \n\r\r\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 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 invalid content-length header size declaration followed by space and null (hex '20 00')	Pass
HTTP/1.1 response with junk string before status header; chunked	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 space+junk string followed by \r\n before first header; chunked	Pass

HTTP/1.1 response with status code 202; 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 response with status code 429; with message-body; chunked	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/1.1\nTransfer-Encoding:chunked; header end \n\n; served chunked	Pass
HTTP/2.0 declared; served chunked	Pass
HTTP/6.-66 declared; served chunked	Pass
HTTP/7.7 declared; served chunked	Pass
Iso	Pass
Kkrunchy	Pass
Kz with high compression using the KZ algorithm (CL=9)	Pass
Nested 7z with high compression using the PPMD algorithm (CL=9) (depth=2)	Pass
Nested 7z with high compression using the PPMD algorithm (CL=9) (depth=3)	Pass
Nested 7z with high compression using the PPMD algorithm (CL=9) (depth=4)	Pass
Nested 7z with high compression using the PPMD algorithm (CL=9) (depth=5)	Pass
Nested Bz2 with high compression (CL=9) (depth=2)	Pass
Nested Bz2 with high compression (CL=9) (depth=3)	Pass
Nested Bz2 with high compression (CL=9) (depth=4)	Pass
Nested Bz2 with high compression (CL=9) (depth=5)	Pass
Nested Gz with high compression (CL=9) (depth=2)	Pass
Nested Gz with high compression (CL=9) (depth=3)	Pass
Nested Gz with high compression (CL=9) (depth=4)	Pass
Nested Gz with high compression (CL=9) (depth=5)	Pass
Nested Zip with high compression using the DEFLATE algorithm (CL=9) (depth=2)	Pass
Nested Zip with high compression using the DEFLATE algorithm (CL=9) (depth=3)	Pass
Nested Zip with high compression using the DEFLATE algorithm (CL=9) (depth=4)	Pass
Nested Zip with high compression using the DEFLATE algorithm (CL=9) (depth=5)	Pass
No status line; chunking indicated; served unchunked	Pass
padded with <5MB	Pass
padded with >25MB	Pass
padded with >25MB and chunked	Pass
padded with >25MB and compressed with deflate	Pass

padded with >25MB and compressed with gzip	Pass
padded with >5MB and <25MB	Pass
padded with >5MB and <25MB and chunked	Pass
padded with >5MB and <25MB and compressed with deflate	Pass
padded with >5MB and <25MB and compressed with gzip	Pass
padded with >5MB and chunked	Pass
padded with 5MB and compressed with deflate	Pass
padded with 5MB and compressed with gzip	Pass
Password protected Kz with high compression using the KZ algorithm (CL=9) (password=password)	Pass
Password protected Rar with high compression using the RAR algorithm (CL=9) (password=password)	Pass
Password protected Rar with high compression using the RAR4 algorithm (CL=9) (password=password)	Pass
Password protected Zip with high compression using the LZMA algorithm (CL=9) (password=password)	Pass
Prefix the status line with (0x0d0a)	Pass
Prefix the status line with (0x0d0a0d0a)	Pass
Prefix the status line with (0x0d0a0d0a0d0a)	Pass
Prefix the status line with (0x20202020)	Pass
Rar with high compression using the RAR algorithm (CL=9)	Pass
Rar with high compression using the RAR4 algorithm (CL=9)	Pass
Relevant headers padded by preceding with hundreds of random custom headers	Pass
Replace the HTTP End of Headers Token with (0x202020200d0a)	Pass
Send download as a ZIP file (bzip2)	Pass
Send download as a ZIP file (bzip2); Add HTTP header (field=X-Padding) (after)	Pass
Send download as a ZIP file (bzip2); HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (0!)	Pass
Send download as a ZIP file (bzip2); HTTP Chunked Transfer Encoding (256-byte)	Pass
Send download as a ZIP file (bzip2); HTTP Identity Content Encoding	Pass
Send download as a ZIP file (bzip2); HTTP Identity Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
Send download as a ZIP file (deflated)	Pass
Send download as a ZIP file (deflated); Add HTTP header (field=X-Padding) (after)	Pass
Send download as a ZIP file (deflated); HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (0!)	Pass
Send download as a ZIP file (deflated); HTTP Chunked Transfer Encoding (256-byte)	Pass
Send download as a ZIP file (deflated); HTTP Identity Content Encoding	Pass
Send download as a ZIP file (deflated); HTTP Identity Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
Send download as a ZIP file (lzma)	Pass
Send download as a ZIP file (lzma); Add HTTP header (field=X-Padding) (after)	Pass

Send download as a ZIP file (lzma); HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (0!)	Pass
Send download as a ZIP file (lzma); HTTP Chunked Transfer Encoding (256-byte)	Pass
Send download as a ZIP file (lzma); HTTP Identity Content Encoding	Pass
Send download as a ZIP file (lzma); HTTP Identity Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
Send download as a ZIP file (none)	Pass
Send download as a ZIP file (none); Add HTTP header (field=X-Padding) (after)	Pass
Send download as a ZIP file (none); HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (0!)	Pass
Send download as a ZIP file (none); HTTP Chunked Transfer Encoding (256-byte)	Pass
Send download as a ZIP file (none); HTTP Identity Content Encoding	Pass
Send download as a ZIP file (none); HTTP Identity Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
Telock	Pass
UPX best	Pass
UPX default	Pass
UPX ultra brute no lzma	Pass
UTF-16 encoding with BOM	Pass
UTF-16 encoding with BOM; no http or html declarations	Pass
UTF-16 encoding with BOM; no http or html declarations; padded with >25MB and chunked	Pass
UTF-16 encoding with BOM; padded with >25MB and chunked	Pass
UTF-16-BE encoding	Pass
UTF-16-BE encoding; no http or html declarations	Pass
UTF-16-LE encoding	Pass
UTF-16-LE encoding; no http or html declarations	Pass
UTF-7 encoding	Pass
UTF-8 encoding	Pass
UTF-8 encoding with BOM	Pass
UTF-8 encoding with BOM; no http or html declarations	Pass
UTF-8 encoding with BOM; no http or html declarations; padded with >25MB and chunked	Pass
UTF-8 encoding with BOM; padded with >25MB and chunked	Pass
UTF-8 encoding; no http or html declarations	Pass
UTF-8 encoding; no http or html declarations; padded with >25MB and chunked	Pass
UTF-8 encoding; padded with >25MB and chunked	Pass
Yoda's Protector	Pass
Yoda's Protector with minimal protections	Pass
Zip with high compression using the DEFLATE algorithm (CL=9)	Pass

Zip with high compression using the LZMA algorithm (CL=9)	Pass
Zip with no compression	Pass

Performance (With Security)			
HTTP Capacity	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 115.6 KB Response	2,363	2,363	3,633.1
2,000 Connections Per Second - 57.4 KB Response	3,242	1,621	1,337.9
4,000 Connections Per Second - 28.0 KB Response	7,201	1,800	924.9
8,000 Connections Per Second - 13.5 KB Response	10,113	1,264	383.7
16,000 Connections Per Second - 6.4 KB Response	12,698	794	280.6
32,000 Connections Per Second - 2.7 KB Response	15,300	478	174.1
HTTPS Capacity (0x13, 0x02)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 113.8 KB Response	1,073	1,073	2,671.6
2,000 Connections Per Second - 54.9 KB Response	1,448	724	1,547.7
4,000 Connections Per Second - 25.7 KB Response	1,688	422	843.6
8,000 Connections Per Second - 11.2 KB Response	1,857	232	0.0
16,000 Connections Per Second - 3.9 KB Response	1,897	119	0.0
32,000 Connections Per Second - 0.2 KB Response	1,899	59	0.0
HTTPS Capacity (0xC0, 0x30)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 115.0 KB Response	1,112	1,112	3,218.2
2,000 Connections Per Second - 56.3 KB Response	1,526	763	1,953.0
4,000 Connections Per Second - 27.0 KB Response	1,842	461	1,006.7
8,000 Connections Per Second - 12.3 KB Response	2,024	253	0.0
16,000 Connections Per Second - 5.0 KB Response	2,029	127	0.0
32,000 Connections Per Second - 1.4 KB Response	2,246	70	0.0

Download Test (With Security)				
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
1 MB Excel OneDrive	Excel	1 MB	0.10	0.40
1 MB Excel Dropbox	Excel	1 MB	0.10	0.40
1 MB Excel Google Drive	Excel	1 MB	0.25	0.40
1 MB Excel HTTP Server	Excel	1 MB	0.30	0.50
1 MB Excel HTTPS Web Server (0x13, 0x02)	Excel	1 MB	0.27	0.40
1 MB Excel HTTPS Web Server (0xC0, 0x30)	Excel	1 MB	0.30	0.40

Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
1 MB PDF OneDrive	PDF	1 MB	0.10	0.40
1 MB PDF Dropbox	PDF	1 MB	0.17	0.40
1 MB PDF Google Drive	PDF	1 MB	0.23	0.40
1 MB PDF HTTP Server	PDF	1 MB	0.27	0.50
1 MB PDF HTTPS Web Server (0x13, 0x02)	PDF	1 MB	0.30	0.40
1 MB PDF HTTPS Web Server (0xC0, 0x30)	PDF	1 MB	0.30	0.40
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
1 MB Word OneDrive	Word	1 MB	0.10	0.40
1 MB Word Dropbox	Word	1 MB	0.10	0.40
1 MB Word Google Drive	Word	1 MB	0.27	0.40
1 MB Word HTTP Server	Word	1 MB	0.27	0.50
1 MB Word HTTPS Web Server (0x13, 0x02)	Word	1 MB	0.27	0.40
1 MB Word HTTPS Web Server (0xC0, 0x30)	Word	1 MB	0.30	0.40
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
1 MB Zip OneDrive	ZIP	1 MB	0.15	0.40
1 MB Zip Dropbox	ZIP	1 MB	0.20	0.40
1 MB Zip Google Drive	ZIP	1 MB	0.25	0.40
1 MB Zip HTTP Server	ZIP	1 MB	0.27	0.50
1 MB Zip HTTPS Web Server (0x13, 0x02)	ZIP	1 MB	0.30	0.40
1 MB Zip HTTPS Web Server (0xC0, 0x30)	ZIP	1 MB	0.30	0.40
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
10 MB Excel OneDrive	Excel	10 MB	0.65	1.65
10 MB Excel Dropbox	Excel	10 MB	0.55	1.10
10 MB Excel Google Drive	Excel	10 MB	0.65	1.25
10 MB Excel HTTP Server	Excel	10 MB	1.13	1.45
10 MB Excel HTTPS Web Server (0x13, 0x02)	Excel	10 MB	1.00	1.35
10 MB Excel HTTPS Web Server (0xC0, 0x30)	Excel	10 MB	0.93	1.30
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
10 MB PDF OneDrive	PDF	10 MB	0.70	1.30
10 MB PDF Dropbox	PDF	10 MB	0.43	1.10
10 MB PDF Google Drive	PDF	10 MB	0.53	1.30
10 MB PDF HTTP Server	PDF	10 MB	1.13	1.40
10 MB HTTPS Web Server (0x13, 0x02)	PDF	10 MB	1.27	1.40
10 MB HTTPS Web Server (0xC0, 0x30)	PDF	10 MB	1.17	1.30

Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
10 MB Word OneDrive	Word	10 MB	0.63	1.15
10 MB Word Dropbox	Word	10 MB	0.43	1.10
10 MB Word Google Drive	Word	10 MB	0.57	1.10
10 MB Word HTTP Server	Word	10 MB	0.93	1.50
10 MB Word HTTPS Web Server (0x13, 0x02)	Word	10 MB	1.43	1.35
10 MB Word HTTPS Web Server (0xC0, 0x30)	Word	10 MB	1.07	1.30
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
10 MB Zip OneDrive	ZIP	10 MB	1.85	1.25
10 MB Zip Dropbox	ZIP	10 MB	0.55	1.10
10 MB Zip Google Drive	ZIP	10 MB	0.80	1.40
10 MB Zip HTTP Server	ZIP	10 MB	1.17	1.45
10 MB Zip HTTPS Web Server (0x13, 0x02)	ZIP	10 MB	0.90	1.40
10 MB Zip HTTPS Web Server (0xC0, 0x30)	ZIP	10 MB	1.20	1.30
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
100 MB Excel OneDrive	Excel	100 MB	6.45	16.50
100 MB Excel Dropbox	Excel	100 MB	2.15	17.00
100 MB Excel Google Drive	Excel	100 MB	5.50	15.50
100 MB Excel HTTP Server	Excel	100 MB	11.07	13.90
100 MB Excel HTTPS Web Server (0x13, 0x02)	Excel	100 MB	10.80	14.15
100 MB Excel HTTPS Web Server (0xC0, 0x30)	Excel	100 MB	8.00	13.00
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
100 MB PDF OneDrive	PDF	100 MB	6.20	11.95
100 MB PDF Dropbox	PDF	100 MB	2.57	11.00
100 MB PDF Google Drive	PDF	100 MB	5.07	13.80
100 MB PDF HTTP Server	PDF	100 MB	10.43	10.85
100 MB PDF HTTPS Web Server (0x13, 0x02)	PDF	100 MB	8.07	10.65
100 MB PDF HTTPS Web Server (0xC0, 0x30)	PDF	100 MB	11.00	12.25
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
100 MB Word OneDrive	Word	100 MB	7.30	18.10
100 MB Word Dropbox	Word	100 MB	2.17	23.00
100 MB Word Google Drive	Word	100 MB	3.67	17.90
100 MB Word HTTP Server	Word	100 MB	7.17	24.35
100 MB Word HTTPS Web Server (0x13, 0x02)	Word	100 MB	9.07	16.60

100 MB Word HTTPS Web Server (0xC0, 0x30)	Word	100 MB	7.67	16.00
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
100 MB Zip OneDrive	ZIP	100 MB	7.15	19.00
100 MB Zip Dropbox	ZIP	100 MB	2.25	17.00
100 MB Zip Google Drive	ZIP	100 MB	4.25	14.75
100 MB Zip HTTP Server	ZIP	100 MB	9.73	14.35
100 MB Zip HTTPS Web Server (0x13, 0x02)	ZIP	100 MB	9.13	13.65
100 MB Zip HTTPS Web Server (0xC0, 0x30)	ZIP	100 MB	8.67	13.05
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
1,000 MB Excel OneDrive	Excel	1,000 MB	64.50	79.00
1,000 MB Excel Dropbox	Excel	1,000 MB	14.50	79.00
1,000 MB Excel Google Drive	Excel	1,000 MB	27.50	41.91
1,000 MB Excel HTTP Server	Excel	1,000 MB	104.33	80.50
1,000 MB Excel HTTPS Web Server (0x13, 0x02)	Excel	1,000 MB	84.67	80.50
1,000 MB Excel HTTPS Web Server (0xC0, 0x30)	Excel	1,000 MB	82.33	80.50
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
1,000 MB PDF OneDrive	PDF	1,000 MB	54.67	80.50
1,000 MB PDF Dropbox	PDF	1,000 MB	17.00	79.00
1,000 MB PDF Google Drive	PDF	1,000 MB	39.50	83.00
1,000 MBPDF HTTP Server	PDF	1,000 MB	94.00	81.00
1,000 MB PDF HTTPS Web Server (0x13, 0x02)	PDF	1,000 MB	82.67	81.50
1,000 MB PDF HTTPS Web Server (0xC0, 0x30)	PDF	1,000 MB	97.67	79.50
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
1,000 MB Word OneDrive	Word	1,000 MB	61.33	81.50
1,000 MB Word Dropbox	Word	1,000 MB	16.67	79.50
1,000 MB Word Google Drive	Word	1,000 MB	55.67	92.00
1,000 MB Word HTTP Server	Word	1,000 MB	90.33	125.50
1,000 MB Word HTTPS Web Server (0x13, 0x02)	Word	1,000 MB	84.33	80.00
1,000 MB Word HTTPS Web Server (0xC0, 0x30)	Word	1,000 MB	96.67	80.50
Description	Type	File Size	Baseline - Time Taken (s)	DUT Time Taken (s)
1,000 MB Zip OneDrive	ZIP	1,000 MB	59.50	82.50
1,000 MB Zip Dropbox	ZIP	1,000 MB	16.50	81.00
1,000 MB Zip Google Drive	ZIP	1,000 MB	18.50	81.00

1,000 MB Zip HTTP Server	ZIP	1,000 MB	90.33	81.50
1,000 MB Zip HTTPS Web Server (0x13, 0x02)	ZIP	1,000 MB	85.00	83.00
1,000 MB Zip HTTPS Web Server (0xC0, 0x30)	ZIP	1,000 MB	94.33	82.00

SPECIAL THANKS

We would like to issue a special thank you to Keysight for providing their CyPerf tool for us to test SSE.

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
515 South Capital of Texas Highway
Suite 225
Austin, TX 78746
info@cyberratings.org
www.cyberratings.org

Please read the disclaimer in this box because it contains important information that binds you. If you do not agree to these conditions, you should not read the rest of this report but should instead return the report immediately to us. "You" or "your" means the person who accesses this report and any entity on whose behalf he/she has obtained this report.

1. The information in this report is subject to change by us without notice, and we disclaim any obligation to update it.

2. The information in this report is believed by us to be accurate and reliable at the time of publication but is not guaranteed. All use of and reliance on this report are at your sole risk. We are not liable or responsible for any damages, losses, or expenses of any nature whatsoever arising from any error or omission in this report.

3. NO WARRANTIES, EXPRESS OR IMPLIED ARE GIVEN BY US. ALL IMPLIED WARRANTIES, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, ARE HEREBY DISCLAIMED AND EXCLUDED BY US. IN NO EVENT SHALL WE BE LIABLE FOR ANY DIRECT, CONSEQUENTIAL, INCIDENTAL, PUNITIVE, EXEMPLARY, OR INDIRECT DAMAGES, OR FOR ANY LOSS OF PROFIT, REVENUE, DATA, COMPUTER PROGRAMS, OR OTHER ASSETS, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

4. This report does not constitute an endorsement, recommendation, or guarantee of any of the products (hardware or software) tested or the hardware and/or software used in testing the products. The testing does not guarantee that there are no errors or defects in the products or that the products will meet your expectations, requirements, needs, or specifications, or that they will operate without interruption.

5. This report does not imply any endorsement, sponsorship, affiliation, or verification by or with any organizations mentioned in this report.

6. All trademarks, service marks, and trade names used in this report are the trademarks, service marks, and trade names of their respective owners.