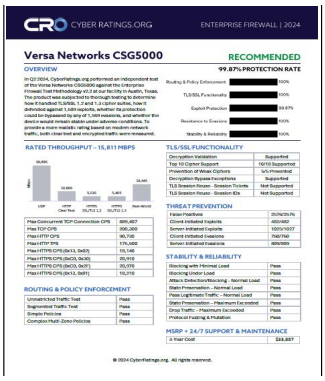

CYBER RATINGS.ORG ENTERPRISE FIREWALL | 2024

Versa Networks CSG5000

RECOMMENDED

99.07% PROTECTION RATE

OVERVIEW

RATED THROUGHPUT – 15,811 MBPS

TLS/SSL FUNCTIONALITY

THREAT PREVENTION

STABILITY & RELIABILITY

ROUTING & POLICY ENFORCEMENT

MSRP + 24/7 SUPPORT & MAINTENANCE

Versa Networks CSG5000

Report

Thank you for downloading this Versa **report**. Carahsoft is the Master Government Aggregator for Versa cyber solutions available via NASA SEWP V, NASPO ValuePoint, and other contract vehicles.

To learn how to take the next step toward acquiring Versa solutions, please check out the following resources and information:

For additional resources:
carah.io/VersaResources

For upcoming events:
carah.io/VersaEvents

For additional Versa solutions:
carah.io/VersaSolutions

For additional cyber solutions:
carah.io/VersaCybersecurity

To set up a meeting:
Versa@carahsoft.com
888-662-2724

To purchase, check out the contract vehicles available for procurement:
carah.io/VersaContracts

Versa Networks CSG5000

RECOMMENDED

OVERVIEW

99.87% PROTECTION RATE

In Q2 2024, CyberRatings.org performed an independent test of the Versa Networks CSG5000 against the Enterprise Firewall Test Methodology v2.2 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 1,509 exploits, whether its protection could be bypassed by any of 1,569 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.

RATED THROUGHPUT – 15,811 MBPS

Max Concurrent TCP Connection CPS	889,407
Max TCP CPS	200,300
Max HTTP CPS	90,730
Max HTTP TPS	175,500
Max HTTPS CPS (0x13, 0x02)	18,140
Max HTTPS CPS (0xC0, 0x30)	20,910
Max HTTPS CPS (0xC0, 0x2F)	20,970
Max HTTPS CPS (0x13, 0x01)	18,210

ROUTING & POLICY ENFORCEMENT

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	Not Supported
TLS Session Reuse - Session IDs	Not Supported

THREAT PREVENTION

False Positives	2576/2576
Client-Initiated Exploits	482/482
Server-Initiated Exploits	1025/1027
Client-Initiated Evasions	760/760
Server-Initiated Evasions	809/809

STABILITY & RELIABILITY

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

MSRP + 24/7 SUPPORT & MAINTENANCE

3-Year Cost	$33,887

Routing & Policy Enforcement

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

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

Routing Functionality	Results
Unrestricted Traffic Test	Pass
Segmented Traffic Test	Pass
Access Control	Results
Simple Policies	Pass
Complex Multi-Zone Policies	Pass

Figure 1 – Routing & Policy Enforcement

TLS/SSL Functionality

The use of the Secure Sockets Layer (SSL) protocol and its current iteration, Transport Layer Security (TLS), 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 device under test (DUT) 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)	Pass
TLS 1.2	11.85%	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)	Pass
TLS 1.2	9.26%	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)	Pass
TLS 1.3	8.07%	TLS_AES_128_GCM_SHA256 (0x13, 0x01)	Pass
TLS 1.2	1.72%	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA8)	Pass
TLS 1.2	0.68%	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28)	Pass
TLS 1.3	0.55%	TLS_CHACHA20_POLY1305_SHA256 (0x13, 0x03)	Pass
TLS 1.2	0.42%	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C)	Pass
TLS 1.2	0.27%	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA9)	Pass
TLS 1.2	0.20%	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B)	Pass

Figure 2 – TLS/SSL Functionality I

First, we tested how the firewall handled cipher suites known to be insecure, using null ciphers (no encryption of data) and anonymous ciphers (no authorization). Then we validated the ability to correctly decrypt and inspect TLS/SSL traffic using prohibited content previously blocked during testing. The content was then encrypted and verified that it was still blocked. We then tested to see if we could permit conditional bypass of decryption. This might be required to preserve privacy for regulatory or other reasons. 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	Not Supported
TLS Session Reuse - Session IDs	Not Supported

Figure 3 – TLS/SSL Functionality II

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

Threat Prevention

A firewall is a mechanism used to protect a trusted network from an untrusted network while allowing authorized communications to pass from one side to the other, thus facilitating secure business use of the Internet. The CyberRatings exploit repository contains exploits demonstrating many protocols and applications. 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.

False Positives

A key to effective protection is correctly identifying and allowing 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 assess the firewall's ability to block attacks while permitting legitimate traffic. If a security product blocked legitimate traffic due to faulty signatures/protection, they were turned off whenever possible.

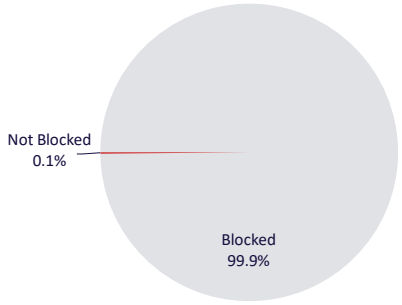
False Positives	Tested	Allowed
Files	1,610	1,610
Sites	966	966

Figure 4 – False Positives

Exploit Protection

99.87% Blocked (1,507/1,509)

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.



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	Number of Exploits Tested	Number of Exploits Blocked
Client-Initiated	482	482
Server-Initiated	1027	1025

Figure 6 – Coverage by Attack Vendor

Coverage by Date

Figure 6 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.

Year	Coverage %
<=2014	100%
2015	100%
2016	100%
2017	99.63%
2018	100%
2019	99.80%
2020	100%
2021	100%
2022	100%
2023	100%

Figure 7 – Coverage by Date

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	100%
Advantech	100%
Apache	100%
Apple	100%
Cisco	100%
HPE	100%
Microsoft	100%
Oracle	100%
Solarwinds	100%
VMware	100%

Figure 8 – Coverage for Top Vendors

Resistance to Evasions

100% Effective (1,569/1,569)

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 which is limited to far fewer exploits.

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

Evasion Technique	Number of Evasions Tested	Number of Evasions Blocked
Client-initiated evasions	760	760
IP Packet Fragmentation	30	30
IP Header	86	86
TCP Transfer Control Block	56	56
TCP Stream Segmentation	88	88
TCP Header	18	18
JavaScript	6	6
HTTP Headers	22	22
HTTP Compression	6	6
HTTP Chunked Encoding	48	48
HTML	16	16
Layered Evasions	384	384
Server-initiated evasions	809	809
IP Packet Fragmentation	96	96
IP Header	199	199
TCP Transfer Control Block	204	204
TCP Stream Segmentation	260	260
TCP Header	50	50

Figure 10 – Evasions by Technique

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 Test	Throughput (Mbps)	Rated Throughput (Mbps)
Raw Packet Processing Performance (UDP Throughput)	38,896	15,811
Plain Text Rated Throughput (Average of HTTP capacity — without delays)	10,886	
HTTPS (TLS/SSL) Rated Throughput (Average of HTTPS Capacity tests)	5,461	
"Real-World" Single Application Flows	18,445	

Figure 11 – 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.

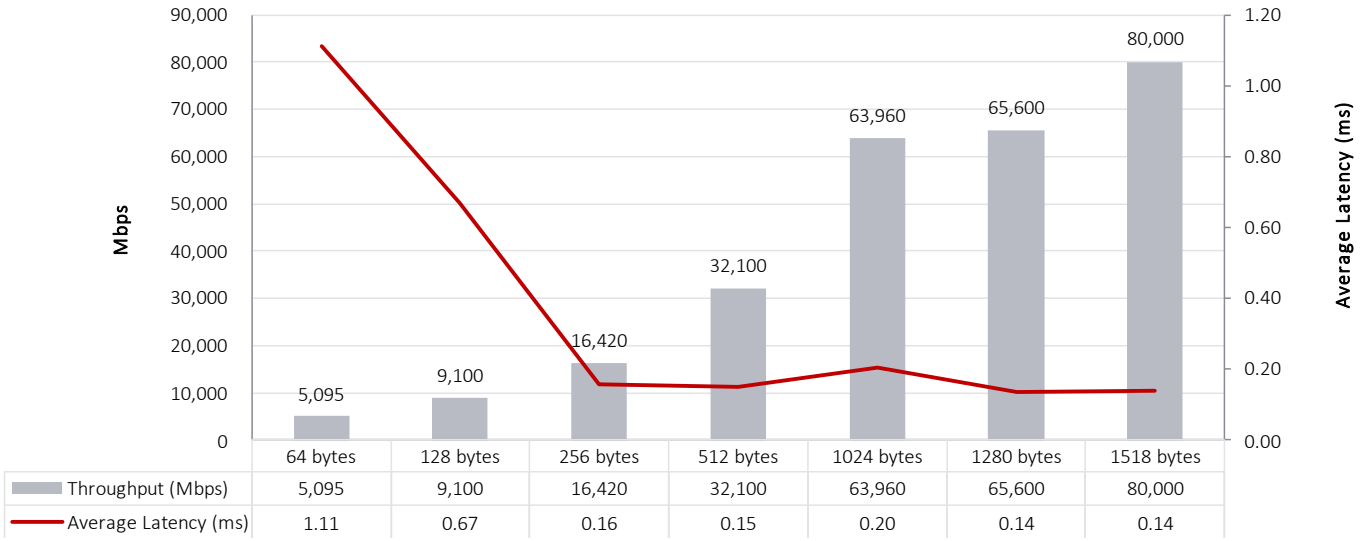
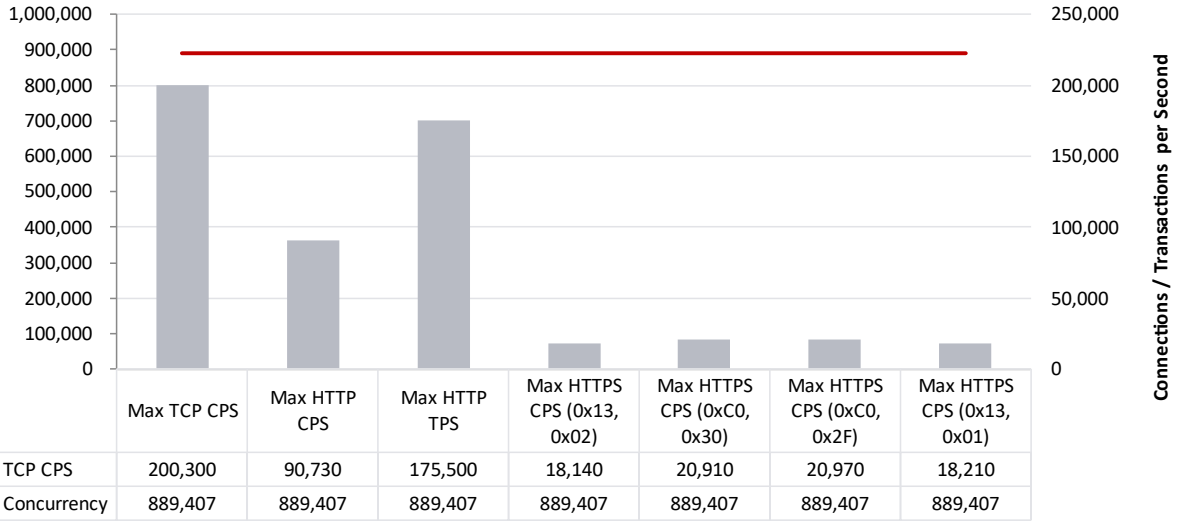


Figure 12 – Raw Packet Processing Performance (UDP Traffic)

Theoretical 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.

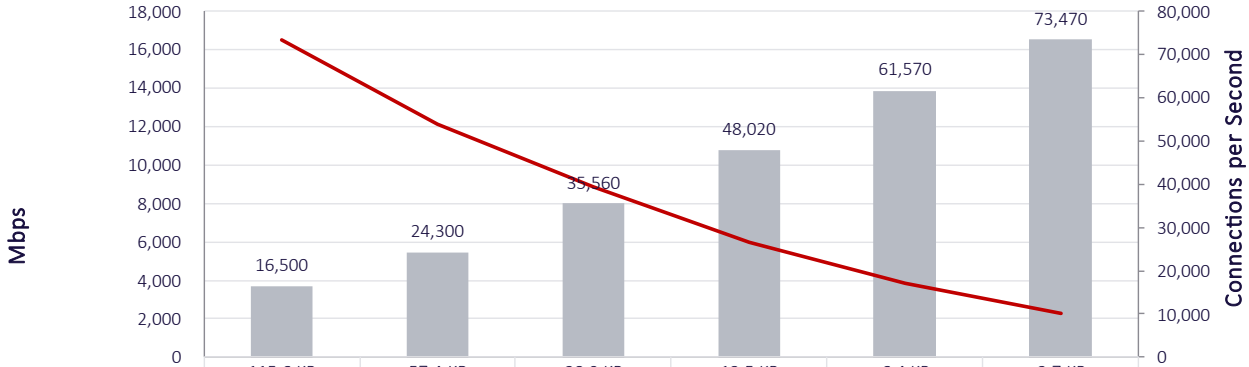


	Max TCP CPS	Max HTTP CPS	Max HTTP TPS	Max HTTPS CPS (0x13, 0x02)	Max HTTPS CPS (0xC0, 0x30)	Max HTTPS CPS (0xC0, 0x2F)	Max HTTPS CPS (0x13, 0x01)
Max TCP CPS	200,300	90,730	175,500	18,140	20,910	20,970	18,210
Max Concurrency	889,407	889,407	889,407	889,407	889,407	889,407	889,407

Figure 13 – 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.



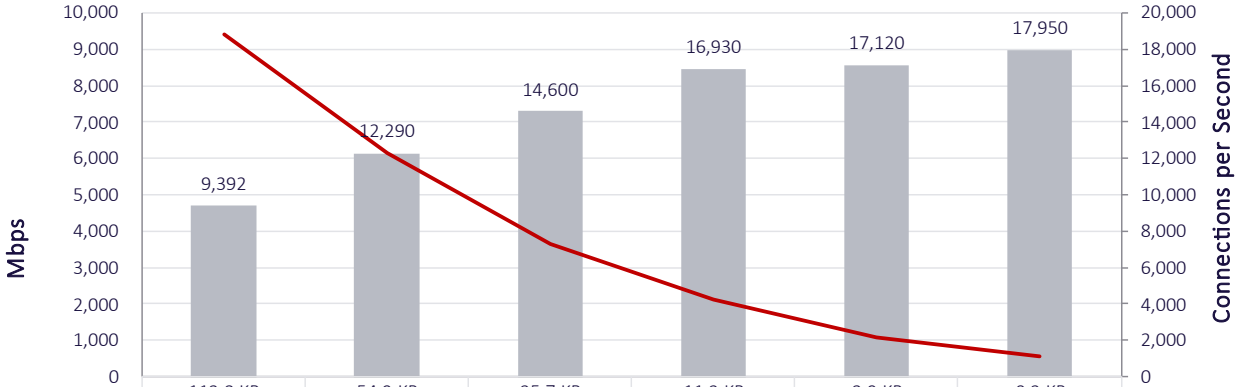
	115.6 KB	57.4 KB	28.0 KB	13.5 KB	6.4 KB	2.7 KB
Connections per Second (CPS)	16,500	24,300	35,560	48,020	61,570	73,470
Throughput (Mbps)	16,500	12,150	8,890	6,003	3,848	2,296
Average Response Time (ms)	1.16	0.93	0.84	0.88	0.87	0.99

Figure 14 – 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).

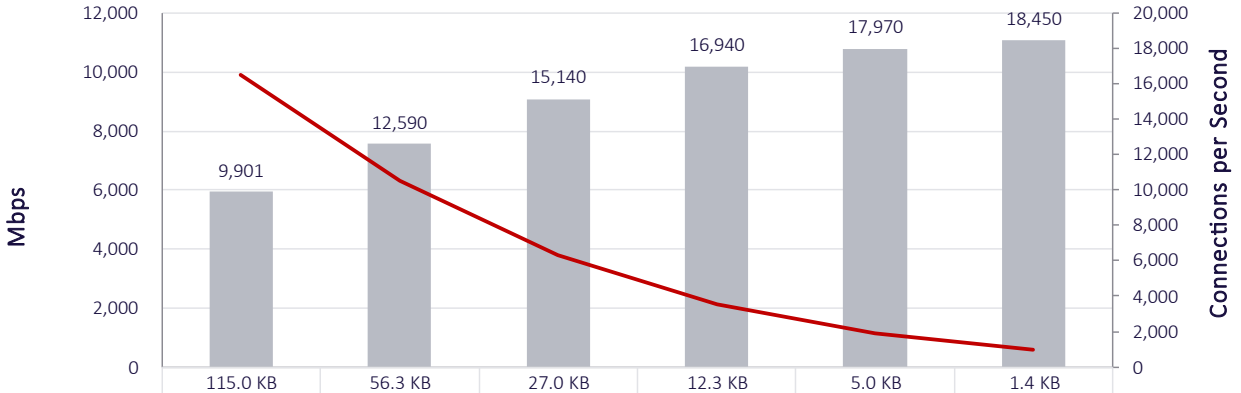
HTTPS Capacity

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



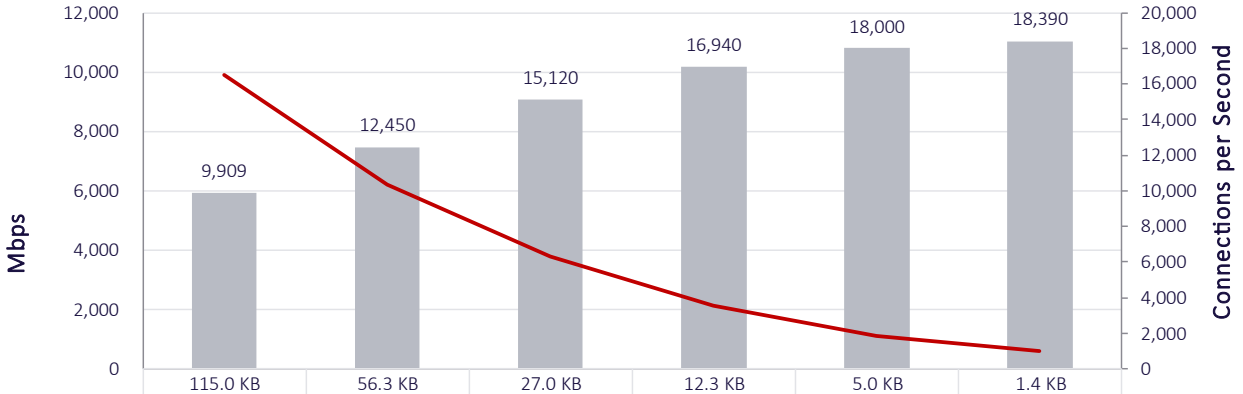
	113.8 KB	54.9 KB	25.7 KB	11.2 KB	3.9 KB	0.2 KB
Connections per Second (CPS)	9,392	12,290	14,600	16,930	17,120	17,950
Throughput (Mbps)	9,392	6,145	3,650	2,116	1,070	561
Average Response Time (ms)	15.06	12.47	10.44	8.02	5.90	5.88

Figure 15 – HTTPS Capacity [TLS_AES_256_GCM_SHA384 (0x13, 0x02)]



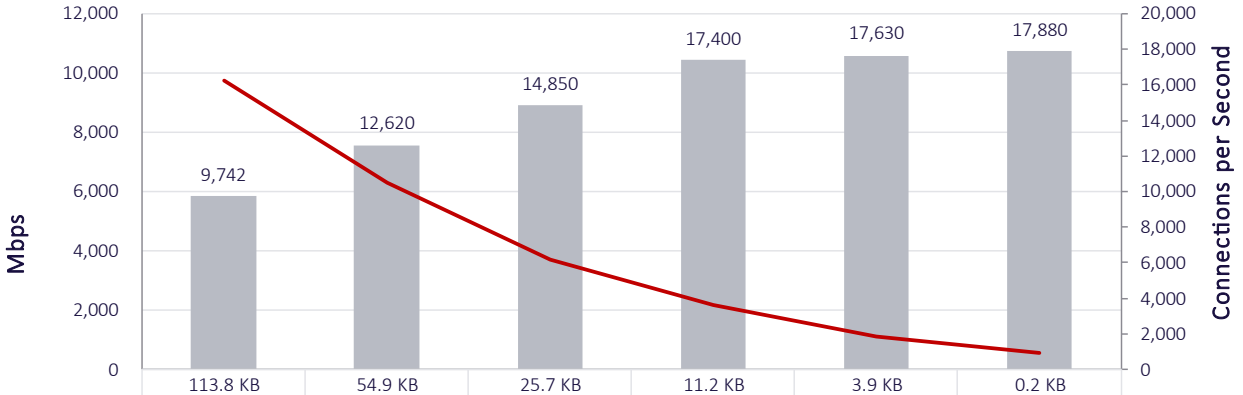
	115.0 KB	56.3 KB	27.0 KB	12.3 KB	5.0 KB	1.4 KB
Connections per Second (CPS)	9,901	12,590	15,140	16,940	17,970	18,450
Throughput (Mbps)	9,901	6,295	3,785	2,118	1,123	577
Average Response Time (ms)	207.03	60.04	8.92	7.58	6.18	6.42

Figure 16 – 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)	9,909	12,450	15,120	16,940	18,000	18,390
Throughput (Mbps)	9,909	6,225	3,780	2,118	1,125	575
Average Response Time (ms)	217.44	61.64	8.80	7.63	6.24	6.18

Figure 17 – 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)	9,742	12,620	14,850	17,400	17,630	17,880
Throughput (Mbps)	9,742	6,310	3,713	2,175	1,102	559
Average Response Time (ms)	17.96	13.29	10.27	8.41	6.21	6.03

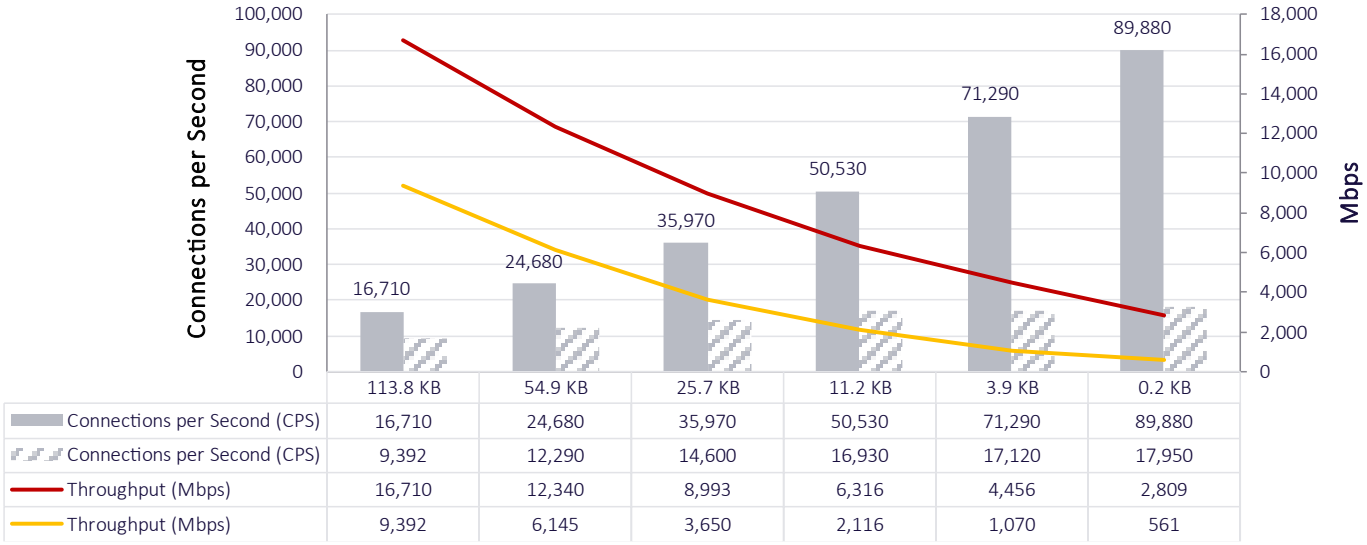
Figure 18 – HTTPS Capacity [TLS_AES_128_GCM_SHA256 (0x13, 0x01)]

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.

Delta between HTTP and HTTPS Capacity & Throughput

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

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

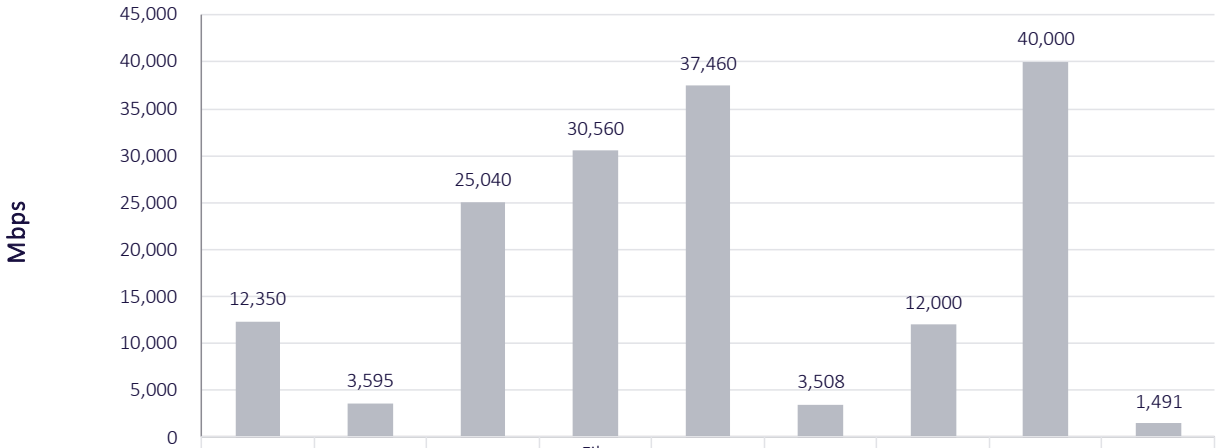


	113.8 KB	54.9 KB	25.7 KB	11.2 KB	3.9 KB	0.2 KB
Connections per Second (CPS)	16,710	24,680	35,970	50,530	71,290	89,880
Connections per Second (CPS)	9,392	12,290	14,600	16,930	17,120	17,950
Throughput (Mbps)	16,710	12,340	8,993	6,316	4,456	2,809
Throughput (Mbps)	9,392	6,145	3,650	2,116	1,070	561

Figure 19 – Delta between HTTP and HTTPS Capacity & Throughput

"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)	12,350	3,595	25,040	30,560	37,460	3,508	12,000	40,000	1,491

Figure 20 – "Real-World" Single Application Flows

Stability & Reliability

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 passed, caused by either the volume of traffic or by the product failing open for any reason, this resulted in a fail.

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 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 test.

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. 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 have been recorded and blocked. If the state tables had been exhausted and the connection was removed from the state table AND failed open (to a bypass condition), the exploit string would not have been reconstructed properly and would not have 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 – Fees paid to the vendor, including software and hardware support, maintenance/updates.
- Implementation Time – Time required to install and configure the DUT in a production environment.
- Upkeep – Time required to apply updates and patches from vendors, including hardware, software, and 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. Price includes Versa Networks Head-End (centralized management) for ten devices at $4,675 for three years.

Enterprise Firewall	Street Price	24/7 Support	Total Cost (1-Year)	Total Cost (3-Years)
Versa Networks CSG5000	$17,500	$5,462	$22,962	$33,887

Figure 22 – 3-Year Cost (USD)

- 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.

Price per Mbps = Price/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 Protection Rate. Using our formula, a device that provides less security, i.e., 50%, will be twice as expensive as a device with 100% security.

Price per Protected Mbps = Price / (Protection Rate x Performance)

(Protection Rate = Routing & Access Control x TLS/SSL / Functionality x Threat Prevention x Stability & Reliability)

Enterprise Firewall	3-Year Cost (Price)	Protection Rate	Rated Throughput	Price per Protected Mbps
Versa Networks CSG5000	$33,887	99.87%	15,811 Mbps	$2.15

Figure 23 – Price per Protected Mbps

Appendix A – Scorecard

Summary			
Vendor		Versa Networks	
Device Model		CSG5000	
Firmware		versa-flexvnf-20240405-041659-5186a33-22.1.4-B	
IPS Version		6446	
Configuration		5 x 10G - 5 port-pairs (limited to 40G)	
Rated Throughput (Max bandwidth 20 Gbps)		15,811	
Routing Functionality		Result	
Unrestricted Traffic Test		Pass	
Segmented Traffic Test		Pass	
Access Control		Result	
Simple Policies		Pass	
Complex Multi-Zone Policies		Pass	
TLS/SSL Support			
Cipher Suites	Prevalence	Version	Result
TLS_AES_256_GCM_SHA384 (0x13, 0x02)	66.51%	TLS 1.3	Pass
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)	11.85%	TLS 1.2	Pass
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)	9.26%	TLS 1.2	Pass
TLS_AES_128_GCM_SHA256 (0x13, 0x01)	8.07%	TLS 1.3	Pass
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA8)	1.72%	TLS 1.2	Pass
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28)	0.68%	TLS 1.2	Pass
TLS_CHACHA20_POLY1305_SHA256 (0x13, 0x03)	0.55%	TLS 1.3	Pass
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C)	0.42%	TLS 1.2	Pass
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC, 0xA9)	0.27%	TLS 1.2	Pass
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B)	0.20%	TLS 1.2	Pass
Null ciphers (no encryption of data)		Version	Result
TLS_RSA_WITH_NULL_MD5 (0x00, 0x01)		SSL 3.0	Pass
TLS_RSA_WITH_NULL_SHA (0x00, 0x02)		SSL 3.0	Pass
Anonymous Ciphers (no authorization)		Version	Result
TLS_DH_anon_WITH_AES_256_CBC_SHA (0x00, 0x3a)		SSL 3.0	Pass
TLS_DH_anon_WITH_RC4_128_MD5 (0x00, 0x18)		SSL 3.0	Pass
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA (0x00, 0x1b)		SSL 3.0	Pass

Decryption Validation			Pass
Decryption Bypass Exceptions			Pass
TLS Session Reuse - Session Tickets			Not Supported
TLS Session Reuse - Session IDs			Not Supported

Threat Prevention	
False Positives	Result
File Download Test	100%
Browsing Test	100%
Exploits	Block Rate
Exploits without Background Network Load	99.87%
Exploits with Background Network Load	99.87%
Evasion Techniques	
IP Header	Result
generic_network_ip_header_001: Interleave Chaff IP Packets (Invalid IP Checksum) (position: before)	Pass
generic_network_ip_header_002: Interleave Chaff IP Packets (Invalid IP Checksum) (position: after)	Pass
generic_network_ip_header_003: Interleave Chaff IP Packets (Invalid IP Checksum) (position: sandwich)	Pass
generic_network_ip_header_004: Interleave Chaff IP Packets (Low TTL Value) (ttl: 1) (position: sandwich)	Pass
generic_network_ip_header_006: Interleave Chaff IP Packets (Invalid/Wrong Protocol Value) (protocol: 0) (position: sandwich)	Pass
generic_network_ip_header_007: Interleave Chaff IP Packets (Invalid/Wrong Protocol Value) (protocol: 255) (position: sandwich)	Pass
generic_network_ip_header_008: Interleave Chaff IP Packets (Invalid/Wrong Protocol Value) (protocol: 150) (position: sandwich)	Pass
generic_network_ip_header_009: Interleave Chaff IP Packets (Invalid IP option) (option: 8307057f00000101) (position: before)	Pass
generic_network_ip_header_010: Interleave Chaff IP Packets (Invalid IP option) (option: 8907057f00000101) (position: before)	Pass
generic_network_ip_header_011: Interleave Chaff IP Packets (Invalid IP option) (option: 83030001) (position: before)	Pass
generic_network_ip_header_012: Interleave Chaff IP Packets (Invalid IP option) (option: 89030001) (position: before)	Pass
generic_network_ip_header_013: Interleave Chaff IP Packets (Invalid IP option) (option: 83070c7f00000101) (position: before)	Pass
generic_network_ip_header_014: Interleave Chaff IP Packets (Invalid IP option) (option: 89070c7f00000101) (position: before)	Pass
generic_network_ip_header_015: Interleave Chaff IP Packets (Invalid IP option) (option: 8307047f0000018307047f0000018307047f000001010101) (position: before)	Pass
generic_network_ip_header_016: Interleave Chaff IP Packets (Invalid IP option) (option: 8907047f0000018907047f0000018907047f000001010101) (position: before)	Pass
generic_network_ip_header_017: Interleave Chaff IP Packets (Invalid IP option) (option: a1da0000) (position: before)	Pass
generic_network_ip_header_018: Interleave Chaff IP Packets (Invalid IP option) (option: 8307057f00000101) (position: after)	Pass

generic_network_ip_header_019: Interleave Chaff IP Packets (Invalid IP option) (option: 8907057f00000101) (position: after)	Pass
generic_network_ip_header_020: Interleave Chaff IP Packets (Invalid IP option) (option: 83030001) (position: after)	Pass
generic_network_ip_header_021: Interleave Chaff IP Packets (Invalid IP option) (option: 89030001) (position: after)	Pass
generic_network_ip_header_022: Interleave Chaff IP Packets (Invalid IP option) (option: 83070c7f00000101) (position: after)	Pass
generic_network_ip_header_023: Interleave Chaff IP Packets (Invalid IP option) (option: 89070c7f00000101) (position: after)	Pass
generic_network_ip_header_024: Interleave Chaff IP Packets (Invalid IP option) (option: 8307047f0000018307047f0000018307047f000001010101) (position: after)	Pass
generic_network_ip_header_025: Interleave Chaff IP Packets (Invalid IP option) (option: 8907047f0000018907047f0000018907047f000001010101) (position: after)	Pass
generic_network_ip_header_026: Interleave Chaff IP Packets (Invalid IP option) (option: a1da0000) (position: after)	Pass
generic_network_ip_header_027: Interleave Chaff IP Packets (Invalid IP option) (option: 8307057f00000101) (position: sandwich)	Pass
generic_network_ip_header_028: Interleave Chaff IP Packets (Invalid IP option) (option: 8907057f00000101) (position: sandwich)	Pass
generic_network_ip_header_029: Interleave Chaff IP Packets (Invalid IP option) (option: 83030001) (position: sandwich)	Pass
generic_network_ip_header_030: Interleave Chaff IP Packets (Invalid IP option) (option: 89030001) (position: sandwich)	Pass
generic_network_ip_header_031: Interleave Chaff IP Packets (Invalid IP option) (option: 83070c7f00000101) (position: sandwich)	Pass
generic_network_ip_header_032: Interleave Chaff IP Packets (Invalid IP option) (option: 89070c7f00000101) (position: sandwich)	Pass
generic_network_ip_header_033: Interleave Chaff IP Packets (Invalid IP option) (option: 8307047f0000018307047f0000018307047f000001010101) (position: sandwich)	Pass
generic_network_ip_header_034: Interleave Chaff IP Packets (Invalid IP option) (option: 8907047f0000018907047f0000018907047f000001010101) (position: sandwich)	Pass
generic_network_ip_header_035: Interleave Chaff IP Packets (Invalid IP option) (option: a1da0000) (position: sandwich)	Pass
windows_network_ip_header_001: Interleave Chaff IP Packets (Invalid IP option) (option: a1040000) (position: before)	Pass
windows_network_ip_header_002: Interleave Chaff IP Packets (Invalid IP option) (option: 4408FF0000000000) (position: before)	Pass
windows_network_ip_header_003: Interleave Chaff IP Packets (Invalid IP option) (option: 440805ff00000000) (position: before)	Pass
windows_network_ip_header_004: Interleave Chaff IP Packets (Invalid IP option) (option: a1040000) (position: before)	Pass
windows_network_ip_header_005: Interleave Chaff IP Packets (Invalid IP option) (option: 4408FF0000000000) (position: before)	Pass
windows_network_ip_header_006: Interleave Chaff IP Packets (Invalid IP option) (option: 440805ff00000000) (position: before)	Pass
windows_network_ip_header_007: Interleave Chaff IP Packets (Invalid IP option) (option: a1040000) (position: before)	Pass
windows_network_ip_header_008: Interleave Chaff IP Packets (Invalid IP option) (option: 4408FF0000000000) (position: before)	Pass
windows_network_ip_header_009: Interleave Chaff IP Packets (Invalid IP option) (option: 440805ff00000000) (position: before)	Pass

IP Packet Fragmentation	Result
generic_network_ip_frag_001: Fragment IP packets (1424-byte)	Pass
generic_network_ip_frag_002: Fragment IP packets (1360-byte)	Pass
generic_network_ip_frag_003: Fragment IP packets (552-byte)	Pass
generic_network_ip_frag_004: Fragment IP packets (248-byte)	Pass
generic_network_ip_frag_005: Fragment IP packets (24-byte)	Pass
generic_network_ip_frag_006: Fragment IP packets (16-byte)	Pass
generic_network_ip_frag_007: Fragment IP packets (8-byte)	Pass
generic_network_ip_frag_008: Fragment IP packets (64-byte); Order packets (reverse)	Pass
generic_network_ip_frag_009: Fragment IP packets (1424-byte); Delay packet (first) (1000 ms)	Pass
generic_network_ip_frag_010: Fragment IP packets (1424-byte); Delay packet (last) (1000 ms)	Pass
generic_network_ip_frag_011: Fragment IP packets (56-byte); Fragment IP packets (24-byte)	Pass
generic_network_ip_frag_012: Fragment IP packets (40-byte); Fragment IP packets (16-byte)	Pass
generic_network_ip_frag_013: Fragment IP packets (80-byte); Fragment IP packets (56-byte); Fragment IP packets (40-byte); Fragment IP packets (32-byte)	Pass
generic_network_ip_frag_014: Fragment IP packets (72-byte); Delay packet (first) (1000 ms); Delay packet (last) (1000 ms)	Pass
generic_network_ip_frag_015: Fragment IP packets (72-byte); Order packets (reverse); Delay packet (first) (1000 ms); Delay packet (last) (1000 ms)	Pass
linux_network_ip_frag_001: Fragment IP packets with partial overlap favoring new (16-byte); Order packets (reverse)	Pass
TCP Header	Result
generic_network_tcp_header_002: Interleave Chaff TCP Segments (Invalid TCP Checksum) (position: after)	Pass
generic_network_tcp_header_008: Interleave Chaff TCP Segments (Unexpected no flags) (position: before)	Pass
generic_network_tcp_header_009: Interleave Chaff TCP Segments (Unexpected no flags) (position: after)	Pass
generic_network_tcp_header_010: Interleave Chaff TCP Segments (Unexpected no flags) (position: sandwich)	Pass
generic_network_tcp_header_011: Interleave Chaff TCP Segments (0 data offset value) (position: before)	Pass
generic_network_tcp_header_012: Interleave Chaff TCP Segments (0 data offset value) (position: after)	Pass
generic_network_tcp_header_013: Interleave Chaff TCP Segments (0 data offset value) (position: sandwich)	Pass
linux_network_tcp_header_001: Interleave Chaff TCP Segments (Older PAWS Timestamps) (position: before)	Pass
linux_network_tcp_header_002: Interleave Chaff TCP Segments (Older PAWS Timestamps) (position: after)	Pass
linux_network_tcp_header_003: Interleave Chaff TCP Segments (Older PAWS Timestamps) (position: sandwich)	Pass
TCP Transfer Control Block	Result
generic_network_tcp_header_004: Interleave Chaff TCP Segments (Out-Of-Window Sequence Numbers) (position: before)	Pass
generic_network_tcp_header_005: Interleave Chaff TCP Segments (Out-Of-Window Sequence Numbers) (position: after)	Pass
generic_network_tcp_header_006: Interleave Chaff TCP Segments (Out-Of-Window Sequence Numbers) (position: sandwich)	Pass

generic_network_tcp_header_007: Interleave Chaff TCP Segments (Retransmission) (position: after)	Pass
generic_network_tcp_tcb_001: Interleave Chaff TCP Segments (Requests to Resync Sequence Numbers Mid-stream) (position: before)	Pass
generic_network_tcp_tcb_002: Interleave Chaff TCP Segments (Requests to Resync Sequence Numbers Mid-stream) (position: after)	Pass
generic_network_tcp_tcb_003: Interleave Chaff TCP Segments (Requests to Resync Sequence Numbers Mid-stream) (position: sandwich)	Pass
generic_network_tcp_tcb_005: Interleave Chaff TCP Segments (Unexpected FIN flag) (position: before)	Pass
generic_network_tcp_tcb_006: Interleave Chaff TCP Segments (Unexpected FIN flag) (position: after)	Pass
generic_network_tcp_tcb_007: Interleave Chaff TCP Segments (Unexpected FIN flag) (position: sandwich)	Pass
generic_network_tcp_tcb_008: Interleave Chaff TCP Segments (Unexpected URG flag) (position: before)	Pass
generic_network_tcp_tcb_009: Interleave Chaff TCP Segments (Unexpected URG flag) (position: after)	Pass
generic_network_tcp_tcb_010: Interleave Chaff TCP Segments (Unexpected URG flag) (position: sandwich)	Pass
generic_network_tcp_tcb_011: Interleave Chaff TCP Segments (Unexpected PSH flag) (position: before)	Pass
generic_network_tcp_tcb_012: Interleave Chaff TCP Segments (Unexpected PSH flag) (position: after)	Pass
generic_network_tcp_tcb_013: Interleave Chaff TCP Segments (Unexpected PSH flag) (position: sandwich)	Pass
generic_network_tcp_tcb_014: Interleave Chaff TCP Segments (Unexpected ECE flag) (position: before)	Pass
generic_network_tcp_tcb_015: Interleave Chaff TCP Segments (Unexpected ECE flag) (position: after)	Pass
generic_network_tcp_tcb_016: Interleave Chaff TCP Segments (Unexpected ECE flag) (position: sandwich)	Pass
generic_network_tcp_tcb_017: Interleave Chaff TCP Segments (Unexpected ECE-CWR-NS flags) (position: before)	Pass
generic_network_tcp_tcb_018: Interleave Chaff TCP Segments (Unexpected ECE-CWR-NS flags) (position: after)	Pass
generic_network_tcp_tcb_019: Interleave Chaff TCP Segments (Unexpected ECE-CWR-NS flags) (position: sandwich)	Pass
generic_network_tcp_tcb_020: Interleave Chaff TCP Segments (Unexpected CWR flag) (position: before)	Pass
generic_network_tcp_tcb_021: Interleave Chaff TCP Segments (Unexpected CWR flag) (position: after)	Pass
generic_network_tcp_tcb_022: Interleave Chaff TCP Segments (Unexpected CWR flag) (position: sandwich)	Pass
generic_network_tcp_tcb_023: Interleave Chaff TCP Segments (Unexpected NS flag) (position: before)	Pass
generic_network_tcp_tcb_024: Interleave Chaff TCP Segments (Unexpected NS flag) (position: after)	Pass
generic_network_tcp_tcb_025: Interleave Chaff TCP Segments (Unexpected NS flag) (position: sandwich)	Pass
linux_network_tcp_tcb_001: Interleave Chaff TCP Segments (Unexpected SYN-FIN flags) (position: before)	Pass
linux_network_tcp_tcb_002: Interleave Chaff TCP Segments (Unexpected SYN-FIN flags) (position: after)	Pass
linux_network_tcp_tcb_003: Interleave Chaff TCP Segments (Unexpected SYN-FIN flags) (position: sandwich)	Pass
linux_network_tcp_tcb_004: Interleave Chaff TCP Segments (Unexpected SYN-ACK-PSH flags) (position: before)	Pass
linux_network_tcp_tcb_005: Interleave Chaff TCP Segments (Unexpected SYN-ACK-PSH flags) (position: after)	Pass
linux_network_tcp_tcb_006: Interleave Chaff TCP Segments (Unexpected SYN-ACK-PSH flags) (position: sandwich)	Pass
TCP Stream Segmentation	Result

generic_network_tcp_seg_001: Segment TCP Segments (1-byte)	Pass
generic_network_tcp_seg_002: Segment TCP Segments (2-byte)	Pass
generic_network_tcp_seg_003: Segment TCP Segments (3-byte)	Pass
generic_network_tcp_seg_004: Segment TCP Segments (4-byte)	Pass
generic_network_tcp_seg_005: Segment TCP Segments (5-byte)	Pass
generic_network_tcp_seg_006: Segment TCP Segments (6-byte)	Pass
generic_network_tcp_seg_007: Segment TCP Segments (7-byte)	Pass
generic_network_tcp_seg_008: Segment TCP Segments (8-byte)	Pass
generic_network_tcp_seg_009: Segment TCP Segments (9-byte)	Pass
generic_network_tcp_seg_010: Segment TCP Segments (10-byte)	Pass
generic_network_tcp_seg_011: Segment TCP Segments (17-byte)	Pass
generic_network_tcp_seg_012: Segment TCP Segments (23-byte)	Pass
generic_network_tcp_seg_013: Segment TCP Segments (31-byte)	Pass
generic_network_tcp_seg_014: Segment TCP Segments (32-byte)	Pass
generic_network_tcp_seg_015: Segment TCP Segments (47-byte)	Pass
generic_network_tcp_seg_016: Segment TCP Segments (69-byte)	Pass
generic_network_tcp_seg_017: Segment TCP Segments (420-byte)	Pass
generic_network_tcp_seg_018: Segment TCP Segments (111-byte)	Pass
generic_network_tcp_seg_019: Segment TCP Segments (123-byte)	Pass
generic_network_tcp_seg_020: Segment TCP Segments (500-byte)	Pass
generic_network_tcp_seg_021: Segment TCP Segments (1024-byte)	Pass
generic_network_tcp_seg_022: Segment TCP Segments (1025-byte)	Pass
generic_network_tcp_seg_023: Segment TCP Segments (1337-byte)	Pass
generic_network_tcp_seg_024: Segment TCP Segments (1400-byte)	Pass
generic_network_tcp_seg_025: Segment TCP Segments (1401-byte)	Pass
generic_network_tcp_seg_026: Segment TCP Segments (30-byte); Order packets (reverse)	Pass
generic_network_tcp_seg_027: Segment TCP Segments (30-byte); Delay packet (first) (1000 ms)	Pass
generic_network_tcp_seg_028: Segment TCP Segments (30-byte); Delay packet (last) (1000 ms)	Pass
generic_network_tcp_seg_029: Segment TCP Segments (512-byte); Delay packet (first) (1000 ms)	Pass
generic_network_tcp_seg_030: Segment TCP Segments (512-byte); Delay packet (last) (1000 ms)	Pass
generic_network_tcp_seg_031: Segment TCP Segments with partial overlap favoring new (3-byte) then no overlap (1-byte)	Pass
generic_network_tcp_seg_032: Segment TCP Segments with partial overlap favoring new (188-byte) then no overlap (54-byte)	Pass
generic_network_tcp_seg_033: Segment TCP Segments (7-byte); Segment TCP Segments (4-byte)	Pass
generic_network_tcp_seg_034: Segment TCP Segments (3-byte); Segment TCP Segments (2-byte)	Pass

generic_network_tcp_seg_035: Segment TCP Segments (10-byte); Segment TCP Segments (7-byte); Segment TCP Segments (5-byte); Segment TCP Segments (4-byte)	Pass
generic_network_tcp_seg_036: Segment TCP Segments (111-byte); Delay packet (first) (1000 ms); Delay packet (last) (1000 ms)	Pass
generic_network_tcp_seg_037: Segment TCP Segments (111-byte); Order packets (reverse); Delay packet (first) (1000 ms); Delay packet (last) (1000 ms)	Pass
linux_network_tcp_seg_001: Segment TCP Segments with partial overlap favoring new (1-byte)	Pass
linux_network_tcp_seg_002: Segment TCP Segments with partial overlap favoring new (2-byte)	Pass
linux_network_tcp_seg_003: Segment TCP Segments with partial overlap favoring new (9-byte)	Pass
linux_network_tcp_seg_004: Segment TCP Segments with partial overlap favoring new (61-byte)	Pass
linux_network_tcp_seg_005: Segment TCP Segments with partial overlap favoring new (420-byte)	Pass
linux_network_tcp_seg_006: Segment TCP Segments (29-byte); Order packets (reverse)	Pass
linux_network_tcp_seg_007: Segment TCP Segments (25-byte); Order packets (reverse)	Pass
linux_network_tcp_seg_008: Segment TCP Segments (17-byte); Order packets (reverse)	Pass
linux_network_tcp_seg_009: Segment TCP Segments (9-byte); Order packets (reverse)	Pass
linux_network_tcp_seg_010: Segment TCP Segments (2-byte); Order packets (reverse)	Pass
linux_network_tcp_seg_011: Segment TCP Segments (1-byte); Order packets (reverse)	Pass
windows_network_tcp_seg_001: Segment TCP Segments with partial overlap favoring old (1-byte)	Pass
windows_network_tcp_seg_002: Segment TCP Segments with partial overlap favoring old (2-byte)	Pass
windows_network_tcp_seg_003: Segment TCP Segments with partial overlap favoring old (9-byte)	Pass
windows_network_tcp_seg_004: Segment TCP Segments with partial overlap favoring old (61-byte)	Pass
windows_network_tcp_seg_005: Segment TCP Segments with partial overlap favoring old (420-byte)	Pass
windows_network_tcp_seg_006: Segment TCP Segments with partial overlap favoring new (15-byte) then partial overlap favoring old (3-byte)	Pass
windows_network_tcp_seg_007: Segment TCP Segments with partial overlap favoring new (3-byte) then partial overlap favoring old (1-byte)	Pass
HTTP Headers	Result
generic_http_header_001: Replace the HTTP End of Headers Token (prefix: 0x202020200d0a)	Pass
generic_http_header_002: Add HTTP header (field: X-Forwarded-For) (value: 127.0.0.1) (position: after)	Pass
generic_http_header_003: Add HTTP header (field: X-Transfer-Encoding) (value: chunked) (position: after)	Pass
generic_http_header_004: Add HTTP header (field: X-Content-Encoding) (value: gzip) (position: after)	Pass
generic_http_header_005: Add HTTP header (field: X-Padding) (value: AAAAAAAAAABBBBBBCCCCCCDDDDDDEEEEEEFFFFFGGGGGZZZZZ...) (position: after)	Pass
generic_http_header_006: Add HTTP header (field: HTTP/1.0) (value: HTTP/1.0) (position: before)	Pass
generic_http_header_007: Add HTTP header (field: X-Test) (value: X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*) (position: before)	Pass
generic_http_header_008: Prefix the status line (prefix: 0x20202020)	Pass
generic_http_header_009: Prefix the status line (prefix: 0x0d0a)	Pass
generic_http_header_010: Prefix the status line (prefix: 0x0d0a0d0a)	Pass
generic_http_header_011: Prefix the status line (prefix: 0x0d0a0d0a0d0a)	Pass

HTTP Chunked Encoding	Result
generic_http_te_001: HTTP Identity Transfer Encoding	Pass
generic_http_te_002: HTTP Chunked Transfer Encoding (1024-byte)	Pass
generic_http_te_003: HTTP Chunked Transfer Encoding (512-byte)	Pass
generic_http_te_004: HTTP Chunked Transfer Encoding (256-byte)	Pass
generic_http_te_005: HTTP Chunked Transfer Encoding (64-byte)	Pass
generic_http_te_006: HTTP Chunked Transfer Encoding (32-byte)	Pass
generic_http_te_007: HTTP Chunked Transfer Encoding (16-byte)	Pass
generic_http_te_008: HTTP Chunked Transfer Encoding (8-byte)	Pass
generic_http_te_009: HTTP Chunked Transfer Encoding (5-byte)	Pass
generic_http_te_010: HTTP Chunked Transfer Encoding (4-byte)	Pass
generic_http_te_011: HTTP Chunked Transfer Encoding (3-byte)	Pass
generic_http_te_012: HTTP Chunked Transfer Encoding (2-byte)	Pass
generic_http_te_013: HTTP Chunked Transfer Encoding (1-byte)	Pass
generic_http_te_014: HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (affix: 0) (position: before)	Pass
generic_http_te_015: HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (affix: 0000000000000000) (position: before)	Pass
generic_http_te_016: HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (affix: 0000000000000000...) (position: before)	Pass
generic_http_te_017: HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (affix: .9) (position: after)	Pass
generic_http_te_018: HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (affix: .999999999999999999) (position: after)	Pass
generic_http_te_019: HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (affix: !!!!) (position: after)	Pass
generic_http_te_020: HTTP Chunked Transfer Encoding (16-byte); Affix to the Chunk Sizes in Non-Terminal HTTP Chunk Headers (affix: !!!!1) (position: after)	Pass
generic_http_te_021: HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (replacement: 000000000)	Pass
generic_http_te_022: HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (replacement: 0+0)	Pass
generic_http_te_023: HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (replacement: 0!)	Pass
generic_http_te_024: HTTP Chunked Transfer Encoding (16-byte); Replace the Chunk Size in the Terminal HTTP Chunk Header (replacement: 0000000000000000...)	Pass
HTTP Compression	Result
generic_http_ce_001: HTTP Identity Content Encoding	Pass
generic_http_ce_002: HTTP Gzip Compression Content Encoding	Pass
generic_http_ce_003: HTTP Deflate Compression Content Encoding	Pass
HTTP Layered	Result
generic_http_multi_001: HTTP Identity Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass

generic_http_multi_002: HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
generic_http_multi_003: HTTP Deflate Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte)	Pass
generic_http_multi_005: HTTP Chunked Transfer Encoding (5-byte); Prefix the status line (prefix: 0x0d0a0d0a)	Pass
generic_http_multi_006: HTTP Chunked Transfer Encoding (5-byte); Prefix the status line (prefix: 0x0d0a)	Pass
generic_http_multi_007: HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (5-byte); Prefix the status line (prefix: 0x0d0a0d0a)	Pass
generic_http_multi_008: HTTP Gzip Compression Content Encoding; HTTP Chunked Transfer Encoding (32-byte); Add HTTP header (field: Transfer-Encoding) (value: identity) (position: after)	Pass
HTML	Result
generic_content_html_001: Add padding to the document (size: 10000 bytes) (padding: A...)	Pass
generic_content_html_002: Add padding to the document (size: 100000 bytes) (padding: A...)	Pass
generic_content_html_003: Add padding to the document (size: 1000000 bytes) (padding: A...)	Pass
generic_content_html_004: Add padding to the document (size: 10000000 bytes) (padding: A...)	Pass
generic_content_html_005: Add padding to the document (size: 10000 bytes) (padding: random)	Pass
generic_content_html_006: Add padding to the document (size: 100000 bytes) (padding: random)	Pass
generic_content_html_007: Add padding to the document (size: 1000000 bytes) (padding: random)	Pass
generic_content_html_008: Add padding to the document (size: 10000000 bytes) (padding: random)	Pass
JavaScript	Result
generic_content_js_001: Add newline padding to each newline in JavaScript (size: 100 newlines)	Pass
generic_content_js_002: Add newline padding to each newline in JavaScript (size: 1000 newlines)	Pass
generic_content_js_003: Add newline padding to each newline in JavaScript (size: 10000 newlines)	Pass
Layered	Result
layered_7ee98093: generic_http_header_005; generic_network_ip_header_015	Pass
layered_979e4068: generic_http_header_005; windows_network_ip_header_002	Pass
layered_516dfb10: generic_http_header_005; generic_network_ip_frag_013	Pass
layered_061b7893: generic_http_header_005; generic_network_tcp_tcb_007	Pass
layered_3f743dba: generic_http_header_005; windows_network_tcp_seg_007	Pass
layered_43ebce86: generic_http_header_005; windows_network_tcp_seg_004	Pass
layered_8c70699b: generic_http_header_005; generic_network_tcp_seg_035	Pass
layered_30d7f4c5: generic_http_te_016; windows_network_ip_header_002	Pass
layered_734228d7: generic_http_te_016; generic_network_ip_header_015	Pass
layered_9c7d4093: generic_http_te_016; generic_network_ip_frag_013	Pass
layered_5951fd45: generic_http_te_016; generic_network_tcp_tcb_007	Pass
layered_14431e80: generic_http_te_016; windows_network_tcp_seg_007	Pass
layered_4a6393c5: generic_http_te_016; windows_network_tcp_seg_004	Pass
layered_8ba6b158: generic_http_te_016; generic_network_tcp_seg_035	Pass

layered_5b5208d8: generic_http_ce_003; generic_network_ip_header_015	Pass
layered_83fb681a: generic_http_ce_003; windows_network_ip_header_002	Pass
layered_b8e47558: generic_http_ce_003; generic_network_ip_frag_013	Pass
layered_d90e0d06: generic_http_ce_003; generic_network_tcp_tcb_007	Pass
layered_0172662e: generic_http_ce_003; generic_network_tcp_seg_035	Pass
layered_4c9ce562: generic_http_ce_003; windows_network_tcp_seg_004	Pass
layered_cd7f49ad: generic_http_ce_003; windows_network_tcp_seg_007	Pass
layered_5550a870: generic_content_html_006; generic_network_ip_header_015	Pass
layered_c037ceee: generic_content_html_002; windows_network_ip_header_002	Pass
layered_c21811cc: generic_content_html_002; generic_network_ip_header_015	Pass
layered_cef05217: generic_content_html_006; windows_network_ip_header_002	Pass
layered_453ed14c: generic_content_html_002; generic_network_ip_frag_013	Pass
layered_d134bc95: generic_content_html_006; generic_network_ip_frag_013	Pass
layered_b5803aa1: generic_content_html_006; generic_network_tcp_tcb_007	Pass
layered_f64a051c: generic_content_html_002; generic_network_tcp_tcb_007	Pass
layered_05ba116d: generic_content_html_002; generic_network_tcp_seg_035	Pass
layered_17c86e3c: generic_content_html_006; windows_network_tcp_seg_007	Pass
layered_9c2fc005: generic_content_html_006; windows_network_tcp_seg_004	Pass
layered_c9ed8329: generic_content_html_002; windows_network_tcp_seg_004	Pass
layered_e6816a11: generic_content_html_006; generic_network_tcp_seg_035	Pass
layered_f96d725a: generic_content_html_002; windows_network_tcp_seg_007	Pass
layered_e7364be0: generic_content_html_002; generic_http_header_005	Pass
layered_f24537f7: generic_content_html_006; generic_http_header_005	Pass
layered_48cd5f7d: generic_content_html_002; generic_http_ce_003	Pass
layered_78eedbb1: generic_content_html_006; generic_http_ce_003	Pass
layered_0b7920e5: generic_content_js_001; windows_network_ip_header_002	Pass
layered_f51ae422: generic_content_js_001; generic_network_ip_header_015	Pass
layered_b7b01f67: generic_content_js_001; generic_network_ip_frag_013	Pass
layered_457dc178: generic_content_js_001; generic_network_tcp_tcb_007	Pass
layered_2df8996a: generic_content_js_001; generic_network_tcp_seg_035	Pass
layered_4b2cb174: generic_content_js_001; windows_network_tcp_seg_007	Pass
layered_fcaf0eab: generic_content_js_001; windows_network_tcp_seg_004	Pass
layered_663d403d: generic_content_js_001; generic_http_header_005	Pass
layered_8b07d07f: generic_content_js_001; generic_http_ce_003	Pass
layered_2be202e8: generic_content_js_001; generic_content_html_002	Pass

layered_aaaafa6d: generic_content_js_001; generic_content_html_006	Pass
layered_2f0b77a9: generic_content_html_006; generic_http_header_005; windows_network_ip_header_002	Pass
layered_53cd1581: generic_content_html_002; generic_http_header_005; windows_network_ip_header_002	Pass
layered_86698159: generic_content_html_002; generic_http_header_005; generic_network_ip_header_015	Pass
layered_e83379f7: generic_content_html_006; generic_http_header_005; generic_network_ip_header_015	Pass
layered_0d4b1b68: generic_content_html_006; generic_http_header_005; generic_network_ip_frag_013	Pass
layered_27ac3095: generic_content_html_002; generic_http_header_005; generic_network_ip_frag_013	Pass
layered_4dad0f19: generic_content_html_002; generic_http_header_005; generic_network_tcp_tcb_007	Pass
layered_9eb068db: generic_content_html_006; generic_http_header_005; generic_network_tcp_tcb_007	Pass
layered_08099802: generic_content_html_006; generic_http_header_005; windows_network_tcp_seg_004	Pass
layered_23fae586: generic_content_html_002; generic_http_header_005; windows_network_tcp_seg_004	Pass
layered_88993c3e: generic_content_html_006; generic_http_header_005; windows_network_tcp_seg_007	Pass
layered_c02b3cf5: generic_content_html_006; generic_http_header_005; generic_network_tcp_seg_035	Pass
layered_e5443d79: generic_content_html_002; generic_http_header_005; windows_network_tcp_seg_007	Pass
layered_f78ff581: generic_content_html_002; generic_http_header_005; generic_network_tcp_seg_035	Pass
layered_4bc7ed9c: generic_content_html_002; generic_http_ce_003; generic_network_ip_header_015	Pass
layered_8fe736d1: generic_content_html_006; generic_http_ce_003; windows_network_ip_header_002	Pass
layered_9d71827a: generic_content_html_006; generic_http_ce_003; generic_network_ip_header_015	Pass
layered_c782fa49: generic_content_html_002; generic_http_ce_003; windows_network_ip_header_002	Pass
layered_2eb8ae10: generic_content_html_002; generic_http_ce_003; generic_network_ip_frag_013	Pass
layered_3a6fc38c: generic_content_html_006; generic_http_ce_003; generic_network_ip_frag_013	Pass
layered_58ec354f: generic_content_html_006; generic_http_ce_003; generic_network_tcp_tcb_007	Pass
layered_80e2f912: generic_content_html_002; generic_http_ce_003; generic_network_tcp_tcb_007	Pass
layered_2208bf00: generic_content_html_006; generic_http_ce_003; windows_network_tcp_seg_004	Pass
layered_87a274d7: generic_content_html_006; generic_http_ce_003; windows_network_tcp_seg_007	Pass
layered_ac9236d7: generic_content_html_002; generic_http_ce_003; generic_network_tcp_seg_035	Pass
layered_cc11f378: generic_content_html_002; generic_http_ce_003; windows_network_tcp_seg_007	Pass
layered_e0b8f45e: generic_content_html_002; generic_http_ce_003; windows_network_tcp_seg_004	Pass
layered_ec77bc7a: generic_content_html_006; generic_http_ce_003; generic_network_tcp_seg_035	Pass
layered_50d9a340: generic_content_js_001; generic_http_header_005; windows_network_ip_header_002	Pass
layered_962ecb36: generic_content_js_001; generic_http_header_005; generic_network_ip_header_015	Pass
layered_9722318f: generic_content_js_001; generic_http_header_005; generic_network_ip_frag_013	Pass
layered_aaccec59: generic_content_js_001; generic_http_header_005; generic_network_tcp_tcb_007	Pass
layered_10183012: generic_content_js_001; generic_http_header_005; windows_network_tcp_seg_004	Pass
layered_242bb0b1: generic_content_js_001; generic_http_header_005; windows_network_tcp_seg_007	Pass

layered_735fa8b3: generic_content_js_001; generic_http_header_005; generic_network_tcp_seg_035	Pass
layered_7d74d71f: generic_content_js_001; generic_http_ce_003; windows_network_ip_header_002	Pass
layered_7fa4f223: generic_content_js_001; generic_http_ce_003; generic_network_ip_header_015	Pass
layered_bfaf4f53: generic_content_js_001; generic_http_ce_003; generic_network_ip_frag_013	Pass
layered_83d55c4d: generic_content_js_001; generic_http_ce_003; generic_network_tcp_tcb_007	Pass
layered_194e29cb: generic_content_js_001; generic_http_ce_003; windows_network_tcp_seg_004	Pass
layered_5bff0da1: generic_content_js_001; generic_http_ce_003; generic_network_tcp_seg_035	Pass
layered_b96dfbf5: generic_content_js_001; generic_http_ce_003; windows_network_tcp_seg_007	Pass
layered_0d2a6bf8: generic_content_js_001; generic_content_html_006; windows_network_ip_header_002	Pass
layered_894a490b: generic_content_js_001; generic_content_html_002; windows_network_ip_header_002	Pass
layered_94660b85: generic_content_js_001; generic_content_html_006; generic_network_ip_header_015	Pass
layered_e019e7ab: generic_content_js_001; generic_content_html_002; generic_network_ip_header_015	Pass
layered_4b4821a7: generic_content_js_001; generic_content_html_002; generic_network_ip_frag_013	Pass
layered_d683c0d6: generic_content_js_001; generic_content_html_006; generic_network_ip_frag_013	Pass
layered_671f59cf: generic_content_js_001; generic_content_html_006; generic_network_tcp_tcb_007	Pass
layered_f30c90f0: generic_content_js_001; generic_content_html_002; generic_network_tcp_tcb_007	Pass
layered_1ec37e60: generic_content_js_001; generic_content_html_006; windows_network_tcp_seg_007	Pass
layered_2489750d: generic_content_js_001; generic_content_html_002; windows_network_tcp_seg_007	Pass
layered_4c5e8817: generic_content_js_001; generic_content_html_006; generic_network_tcp_seg_035	Pass
layered_805155c3: generic_content_js_001; generic_content_html_002; generic_network_tcp_seg_035	Pass
layered_8534c3c1: generic_content_js_001; generic_content_html_006; windows_network_tcp_seg_004	Pass
layered_af95f69f: generic_content_js_001; generic_content_html_002; windows_network_tcp_seg_004	Pass
layered_c657e125: generic_content_js_001; generic_content_html_002; generic_http_header_005	Pass
layered_d2c910e7: generic_content_js_001; generic_content_html_006; generic_http_header_005	Pass
layered_12ac4e4f: generic_content_js_001; generic_content_html_006; generic_http_ce_003	Pass
layered_73e64ec9: generic_content_js_001; generic_content_html_002; generic_http_ce_003	Pass
layered_e4321334: generic_http_multi_007; windows_network_ip_header_002	Pass
layered_f4b43f6c: generic_http_multi_007; generic_network_ip_header_015	Pass
layered_c3f5e32e: generic_http_multi_007; generic_network_ip_frag_013	Pass
layered_9c92647d: generic_http_multi_007; generic_network_tcp_tcb_007	Pass
layered_50b514a1: generic_http_multi_007; windows_network_tcp_seg_004	Pass
layered_531604fe: generic_http_multi_007; windows_network_tcp_seg_007	Pass
layered_f50c24f2: generic_http_multi_007; generic_network_tcp_seg_035	Pass
layered_4e7d8aad: generic_content_html_006; generic_http_multi_007	Pass
layered_e447c8c3: generic_content_html_002; generic_http_multi_007	Pass

layered_410523d3: generic_content_js_001; generic_http_multi_007	Pass
layered_4d8c70fb: generic_content_js_001; generic_content_html_006; generic_http_header_005; generic_network_ip_header_015	Pass
layered_58f84233: generic_content_js_001; generic_content_html_006; generic_http_header_005; windows_network_ip_header_002	Pass
layered_7197f8f7: generic_content_js_001; generic_content_html_002; generic_http_header_005; generic_network_ip_header_015	Pass
layered_f8b3ae43: generic_content_js_001; generic_content_html_002; generic_http_header_005; windows_network_ip_header_002	Pass
layered_393cf805: generic_content_js_001; generic_content_html_006; generic_http_header_005; generic_network_ip_frag_013	Pass
layered_d392ef2f: generic_content_js_001; generic_content_html_002; generic_http_header_005; generic_network_ip_frag_013	Pass
layered_2f5ce065: generic_content_js_001; generic_content_html_006; generic_http_header_005; generic_network_tcp_tcb_007	Pass
layered_5298371b: generic_content_js_001; generic_content_html_002; generic_http_header_005; generic_network_tcp_tcb_007	Pass
layered_62ed1320: generic_content_js_001; generic_content_html_006; generic_http_header_005; windows_network_tcp_seg_007	Pass
layered_73f585c0: generic_content_js_001; generic_content_html_002; generic_http_header_005; windows_network_tcp_seg_004	Pass
layered_8ee8eed3: generic_content_js_001; generic_content_html_006; generic_http_header_005; generic_network_tcp_seg_035	Pass
layered_b074d732: generic_content_js_001; generic_content_html_002; generic_http_header_005; generic_network_tcp_seg_035	Pass
layered_efbe588e: generic_content_js_001; generic_content_html_006; generic_http_header_005; windows_network_tcp_seg_004	Pass
layered_fcb99d20: generic_content_js_001; generic_content_html_002; generic_http_header_005; windows_network_tcp_seg_007	Pass
layered_4e71ff6e: generic_content_js_001; generic_content_html_006; generic_http_ce_003; generic_network_ip_header_015	Pass
layered_95adb982: generic_content_js_001; generic_content_html_002; generic_http_ce_003; generic_network_ip_header_015	Pass
layered_b19c8f79: generic_content_js_001; generic_content_html_006; generic_http_ce_003; windows_network_ip_header_002	Pass
layered_e5ef51bb: generic_content_js_001; generic_content_html_002; generic_http_ce_003; windows_network_ip_header_002	Pass
layered_278f606a: generic_content_js_001; generic_content_html_006; generic_http_ce_003; generic_network_ip_frag_013	Pass
layered_3d86ca8e: generic_content_js_001; generic_content_html_002; generic_http_ce_003; generic_network_ip_frag_013	Pass
layered_5afa3d2d: generic_content_js_001; generic_content_html_002; generic_http_ce_003; generic_network_tcp_tcb_007	Pass
layered_b05928b3: generic_content_js_001; generic_content_html_006; generic_http_ce_003; generic_network_tcp_tcb_007	Pass
layered_15f77702: generic_content_js_001; generic_content_html_006; generic_http_ce_003; windows_network_tcp_seg_004	Pass
layered_55d9c305: generic_content_js_001; generic_content_html_002; generic_http_ce_003; generic_network_tcp_seg_035	Pass
layered_7fc67291: generic_content_js_001; generic_content_html_006; generic_http_ce_003; windows_network_tcp_seg_007	Pass

layered_8cdf5cd8: generic_content_js_001; generic_content_html_006; generic_http_ce_003; generic_network_tcp_seg_035	Pass
layered_c7ab2156: generic_content_js_001; generic_content_html_002; generic_http_ce_003; windows_network_tcp_seg_004	Pass
layered_d036f238: generic_content_js_001; generic_content_html_002; generic_http_ce_003; windows_network_tcp_seg_007	Pass
layered_1ffac54a: generic_content_html_002; generic_http_multi_007; windows_network_ip_header_002	Pass
layered_62426459: generic_content_html_006; generic_http_multi_007; generic_network_ip_header_015	Pass
layered_6644ad1c: generic_content_html_006; generic_http_multi_007; windows_network_ip_header_002	Pass
layered_b8486f5d: generic_content_html_002; generic_http_multi_007; generic_network_ip_header_015	Pass
layered_3860cf3a: generic_content_html_002; generic_http_multi_007; generic_network_ip_frag_013	Pass
layered_e182df2c: generic_content_html_006; generic_http_multi_007; generic_network_ip_frag_013	Pass
layered_81e50026: generic_content_html_006; generic_http_multi_007; generic_network_tcp_tcb_007	Pass
layered_8b423360: generic_content_html_002; generic_http_multi_007; generic_network_tcp_tcb_007	Pass
layered_175fc76a: generic_content_html_002; generic_http_multi_007; windows_network_tcp_seg_007	Pass
layered_5d0875e3: generic_content_html_006; generic_http_multi_007; windows_network_tcp_seg_007	Pass
layered_6266832d: generic_content_html_006; generic_http_multi_007; windows_network_tcp_seg_004	Pass
layered_6cc8da4c: generic_content_html_006; generic_http_multi_007; generic_network_tcp_seg_035	Pass
layered_71bd5def: generic_content_html_002; generic_http_multi_007; windows_network_tcp_seg_004	Pass
layered_99aad529: generic_content_html_002; generic_http_multi_007; generic_network_tcp_seg_035	Pass
layered_131a173b: generic_content_js_001; generic_http_multi_007; generic_network_ip_header_015	Pass
layered_4be064bf: generic_content_js_001; generic_http_multi_007; windows_network_ip_header_002	Pass
layered_1628f3c9: generic_content_js_001; generic_http_multi_007; generic_network_ip_frag_013	Pass
layered_f4cb19df: generic_content_js_001; generic_http_multi_007; generic_network_tcp_tcb_007	Pass
layered_67dc95a9: generic_content_js_001; generic_http_multi_007; windows_network_tcp_seg_004	Pass
layered_8cced2f0: generic_content_js_001; generic_http_multi_007; generic_network_tcp_seg_035	Pass
layered_ef751204: generic_content_js_001; generic_http_multi_007; windows_network_tcp_seg_007	Pass
layered_205f66d5: generic_content_js_001; generic_content_html_002; generic_http_multi_007	Pass
layered_e26c961e: generic_content_js_001; generic_content_html_006; generic_http_multi_007	Pass
layered_105c35e1: generic_content_js_001; generic_content_html_006; generic_http_multi_007; windows_network_ip_header_002	Pass
layered_464b075e: generic_content_js_001; generic_content_html_002; generic_http_multi_007; generic_network_ip_header_015	Pass
layered_655532ce: generic_content_js_001; generic_content_html_002; generic_http_multi_007; windows_network_ip_header_002	Pass
layered_b197265f: generic_content_js_001; generic_content_html_006; generic_http_multi_007; generic_network_ip_header_015	Pass
layered_6115783f: generic_content_js_001; generic_content_html_002; generic_http_multi_007; generic_network_ip_frag_013	Pass
layered_dc573d5a: generic_content_js_001; generic_content_html_006; generic_http_multi_007; generic_network_ip_frag_013	Pass

layered_62aeaddc: generic_content_js_001; generic_content_html_006; generic_http_multi_007; generic_network_tcp_tcb_007	Pass
layered_bc7a7a51: generic_content_js_001; generic_content_html_002; generic_http_multi_007; generic_network_tcp_tcb_007	Pass
layered_25338d3d: generic_content_js_001; generic_content_html_006; generic_http_multi_007; windows_network_tcp_seg_007	Pass
layered_43d1b4a8: generic_content_js_001; generic_content_html_006; generic_http_multi_007; generic_network_tcp_seg_035	Pass
layered_59be600a: generic_content_js_001; generic_content_html_006; generic_http_multi_007; windows_network_tcp_seg_004	Pass
layered_73022ea3: generic_content_js_001; generic_content_html_002; generic_http_multi_007; windows_network_tcp_seg_007	Pass
layered_b898fd82: generic_content_js_001; generic_content_html_002; generic_http_multi_007; generic_network_tcp_seg_035	Pass
layered_eb06be65: generic_content_js_001; generic_content_html_002; generic_http_multi_007; windows_network_tcp_seg_004	Pass

Performance			
Raw Packet Processing Performance (UDP Throughput)		Throughput (Mbps)	Latency (ms)
64 Byte Frames		5,095	1.11
128 Byte Frames		9,100	0.67
256 Byte Frames		16,420	0.16
512 Byte Frames		32,100	0.15
1024 Byte Frames		63,960	0.20
1280 Byte Frames		65,600	0.14
1518 Byte Frames		80,000	0.14
Maximum Capacity	CPS	TPS	
Max Concurrent TCP Connection	889,407	-	
Max TCP CPS	200,300	-	
Max HTTP CPS	90,730	-	
Max HTTP TPS	-	175,500	
Max HTTPS CPS (0x13, 0x01)	18,140	-	
Max HTTPS CPS (0x13, 0x02)	20,910	-	
Max HTTPS CPS (0xC0, 0x2F)	20,970	-	
Max HTTPS CPS (0xC0, 0x30)	18,210	-	
HTTP Capacity	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 115.6 KB Response	16,500	16,500	1.16
2,000 Connections Per Second - 57.4 KB Response	24,300	12,150	0.93
4,000 Connections Per Second - 28.0 KB Response	35,560	8,890	0.84
8,000 Connections Per Second - 13.5 KB Response	48,020	6,003	0.88

16,000 Connections Per Second - 6.4 KB Response	61,570	3,848	0.87
32,000 Connections Per Second - 2.7 KB Response	73,470	2,296	0.99
HTTPS Capacity (0x13, 0x02)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 113.8 KB Response	9,392	9,392	15.06
2,000 Connections Per Second - 54.9 KB Response	12,290	6,145	12.47
4,000 Connections Per Second - 25.7 KB Response	14,600	3,650	10.44
8,000 Connections Per Second - 11.2 KB Response	16,930	2,116	8.02
16,000 Connections Per Second - 3.9 KB Response	17,120	1,070	5.90
32,000 Connections Per Second - 0.2 KB Response	17,950	561	5.88
HTTPS Capacity (0xC0, 0x30)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 115.0 KB Response	9,901	9,901	207.03
2,000 Connections Per Second - 56.3 KB Response	12,590	6,295	60.04
4,000 Connections Per Second - 27.0 KB Response	15,140	3,785	8.92
8,000 Connections Per Second - 12.3 KB Response	16,940	2,118	7.58
16,000 Connections Per Second - 5.0 KB Response	17,970	1,123	6.18
32,000 Connections Per Second - 1.4 KB Response	18,450	577	6.42
HTTPS Capacity (0xC0, 0x2F)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 115.0 KB Response	9,909	9,909	217.44
2,000 Connections Per Second - 56.3 KB Response	12,450	6,225	61.64
4,000 Connections Per Second - 27.0 KB Response	15,120	3,780	8.80
8,000 Connections Per Second - 12.3 KB Response	16,940	2,118	7.63
16,000 Connections Per Second - 5.0 KB Response	18,000	1,125	6.24
32,000 Connections Per Second - 1.4 KB Response	18,390	575	6.18
HTTPS Capacity (0x13, 0x01)	CPS	Throughput (Mbps)	Response Time (ms)
1,000 Connections Per Second - 113.8 KB Response	9,742	9,742	17.96
2,000 Connections Per Second - 54.9 KB Response	12,620	6,310	13.29
4,000 Connections Per Second - 25.7 KB Response	14,850	3,713	10.27
8,000 Connections Per Second - 11.2 KB Response	17,400	2,175	8.41
16,000 Connections Per Second - 3.9 KB Response	17,630	1,102	6.21
32,000 Connections Per Second - 0.2 KB Response	17,880	559	6.03
Delta between HTTP & HTTPS (TLS 1.3) Capacity	CPS	Throughput (Mbps)	Response Time (ms)
HTTP Capacity			
1,000 Connections Per Second - 113.8 KB Response	16,710	16,710	0.95
2,000 Connections Per Second - 54.9 KB Response	24,680	12,340	0.87
4,000 Connections Per Second - 25.7 KB Response	35,970	8,993	0.93

8,000 Connections Per Second - 11.2 KB Response	50,530	6,316	0.98
16,000 Connections Per Second - 3.9 KB Response	71,290	4,456	0.99
32,000 Connections Per Second - 0.2 KB Response	89,880	2,809	1.16
HTTPS Capacity (0x13, 0x02)			
1,000 Connections Per Second - 113.8 KB Response	9,392	9,392	15.06
2,000 Connections Per Second - 54.9 KB Response	12,290	6,145	12.47
4,000 Connections Per Second - 25.7 KB Response	14,600	3,650	10.44
8,000 Connections Per Second - 11.2 KB Response	16,930	2,116	8.02
16,000 Connections Per Second - 3.9 KB Response	17,120	1,070	5.90
32,000 Connections Per Second - 0.2 KB Response	17,950	561	5.88
Real-World Single Application Flows		Throughput (Mbps)	
Telephony (SIP)		12,350	
Finance (FIX)		3,595	
Mail (SMTP)		25,040	
File Transfer (FTP)		30,560	
File Server (SMB)		37,460	
Remote (RDP)		3,508	
Video (Youtube)		12,000	
Database (MSSQL)		40,000	
Web Conference (Webex)		1,491	
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 B – CyberRatings Rating Matrix

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

SPECIAL THANKS

We would like to issue a special thank you to Keysight for providing their CyPerf and Breaking Point tools for us to test 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
515 South Capital of Texas Highway
Suite 225
Austin, TX 78746
info@cyberratings.org
www.cyberratings.org