



NEXT GENERATION FIREWALL TEST REPORT

Palo Alto Networks PA-5220 PAN-OS 8.1.6-h2

JULY 17, 2019

Authors – Devon James, Keith Bormann, Julian Owusu-Abrokwa

Palo Alto Networks PA-5220 PAN-OS 8.1.6-h2		
Summary	In 1H 2019, NSS Labs performed an independent test of the Palo Alto Networks PA-5220 PAN-OS 8.1.6-h2. This report focuses on the main differentiators for next generation firewall (NGFW) products: Security, Performance/Functionality, and Cost.	
Security	The ideal NGFW delivers a high block rate for exploits and evasions while at the same time providing the expected stability and reliability.	
	Security Effectiveness	
	Exploit Block Rate ¹	97.87%
	Evasions Blocked	406/406
	Stability & Reliability	PASS
Performance/Functionality	The ideal NGFW provides high performance for both encrypted and non-encrypted traffic while offering support for the most used ciphers.	
	Performance	
	HTTP NSS -Tested Throughput	7,737 Mbps
	HTTPS (SSL/TLS) NSS-Tested Throughput	1,913 Mbps
	Combined HTTP/HTTPS NSS-Tested Throughput	3,660 Mbps
	SSL/TLS Functionality	
	Decryption validation	PASS
	Top 30 cipher support ²	20/24
	Emergent ciphers Support for x25519 Elliptical curve specification	0/2 FAIL
	Prevention of weak ciphers	PASS
	Decryption bypass policy based on Layer 3 information	PASS
	Decryption bypass policy based on Layer 4 information	PASS
	Decryption bypass policy based on Server Name Indication (SNI) TLS extension information	PASS
	Decryption bypass policy based on Common Name (CN) and/or Subject Alternative Name (SAN)	PASS
Certificate validation	PASS	
Support for session method: ID reuse ticket reuse	PASS PASS	
Cost	The ideal NGFW is scalable, delivers continuous uptime, and has low maintenance and support costs.	
	Total Cost of Ownership (TCO)	
	3-Year TCO (US\$)	\$70,500
The product was subjected to thorough testing based on the Next Generation Firewall Test Methodology v9.0, the Secure Sockets Layer/Transport Layer Security Performance Test Methodology v1.3, and the Evasions Test Methodology v1.1 (www.nsslabs.com). As with any NSS Labs group test, the test described in this report was conducted free of charge.		

¹ Exploit block rate is defined as the number of live exploits, exploits from the *NSS Labs Exploit Library*, and script obfuscations blocked under test.

² For additional details, see the SSL/TLS Functionality section.

Table of Contents

Security Effectiveness 5

- False Positive Testing*5
- NSS Labs Exploit Library5
 - Coverage by Attack Vector*6
 - Coverage by Impact Type*.....6
 - Coverage by Date*.....7
 - Coverage by Target Vendor*7
- Script Obfuscation8
- Live Exploits8
- Resistance to Evasion Techniques8

Performance 10

- Raw Packet Processing Performance (UDP Throughput)10
- Raw Packet Processing Performance (UDP Latency)11
- Maximum Capacity12
- HTTP Capacity13
- Application Average Response Time – HTTP13
- HTTP Capacity with HTTP Persistent Connections14
- HTTP Capacity with Bidirectional HTTP Persistent Connections15
- Single Application Flows15
- HTTP NSS-Tested Throughput16

SSL/TLS 17

- SSL/TLS Functionality17
 - Decryption Validation*17
 - Cipher Selection*17
 - Cipher Support*17
 - Top 30 Cipher Suites from the Alexa Top 1 Million*18
 - Support for Emergent Ciphers*.....18
 - Support for x25519 Elliptic Curve*.....18
 - Prevention of Weak Ciphers*.....18
 - Decryption Bypass Exceptions*.....18
 - Certificate Validation*19
 - TLS Session Reuse*.....19
- Maximum SSL/TLS Handshakes per Second20
- HTTPS Throughput Capacity21
- HTTPS (SSL/TLS) NSS-Tested Throughput24

Stability and Reliability 25

Total Cost of Ownership (TCO)..... 26

- Installation Hours26
- Total Cost of Ownership27

Appendix A: Product Scorecard..... 28

Test Methodology 49

Contact Information 49

Table of Figures

Figure 1 – NSS Labs Exploit Library Attacks Blocked (%)	5
Figure 2 – Coverage by Attack Vector	6
Figure 3 – Product Coverage by Date	7
Figure 4 – Product Coverage by Target Vendor	7
Figure 5 – Script Obfuscation Attacks Blocked (%)	8
Figure 6 – Live Attacks Blocked (%)	8
Figure 7 – Resistance to Evasion Results	9
Figure 8 – Raw Packet Processing Performance (UDP Traffic)	10
Figure 9 – UDP Latency in Microseconds.....	11
Figure 10 – Concurrency and Connection Rates.....	12
Figure 11 – HTTP Capacity	13
Figure 12 – Average Application Response Time (Milliseconds)	13
Figure 13 – HTTP Capacity with HTTP Persistent Connections	14
Figure 14 – HTTP Capacity with Bidirectional HTTP Persistent Connections.....	15
Figure 15 – Single Application Flows	15
Figure 16 – HTTP NSS-Tested Throughput (Mbps)	16
Figure 17 – SSL/TLS Functionality Tested	19
Figure 18 – Maximum HTTP(S) Connections per Second	20
Figure 19 – HTTP Capacity (No Persistence) Single HTTP GET Request (2,880 KB)	21
Figure 20 – HTTP Capacity (No Persistence) Single HTTP GET Request (768 KB)	21
Figure 21 – HTTP Capacity (No Persistence) Single HTTP GET Request (192 KB)	22
Figure 22 – HTTP Capacity (No Persistence) Single HTTP GET Request (44 KB)	22
Figure 23 – HTTP Capacity with Persistent Connections with 10 HTTP GET Requests (288 KB).....	22
Figure 24 – HTTP Capacity with Persistent Connections with 10 HTTP GET Requests (76.8 KB).....	23
Figure 25 – HTTP Capacity with Persistent Connections with 10 HTTP GET Requests (19.2KB)	23
Figure 26 – HTTP Capacity with Persistent Connections with 10 HTTP GET Requests (4.4 KB).....	23
Figure 27 – HTTPS (SSL/TLS) NSS-Tested Throughput (Mbps).....	24
Figure 28 – Stability and Reliability Results	25
Figure 29 – Sensor Installation Time (Hours).....	26
Figure 30 –3-Year TCO (US\$)	27
Figure 31 – Detailed Scorecard.....	48

Security Effectiveness

The firewall market is one of the largest and most mature security markets. Firewalls have undergone several stages of development, from early packet filtering and circuit relay firewalls to application-layer (proxy-based) and dynamic packet filtering firewalls. Throughout their history, however, the goal has been to enforce an access control policy between two networks, and they should therefore be viewed as an implementation of policy.

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. With the emergence of HTML 5, web browsers, and security threats, however, firewalls are evolving further. NGFWs traditionally have been deployed to defend the network on the edge, but some enterprises have expanded their deployment to include internal segmentation.

As Web 3.0 trends push critical business applications through firewall ports that previously were reserved for a single function, such as HTTP, legacy firewall technology is effectively blinded. It is unable to differentiate between actual HTTP traffic and non-HTTP services tunneling over port 80, such as VoIP or instant messaging. Today, application-level monitoring must be performed in addition to analysis of port and destination. Firewalls are evolving to address this increased complexity.

It is no longer possible to rely on port and protocol combinations alone to define network applications. The NGFW must be capable of determining which applications are running, regardless of which ports they are using, in order to secure them effectively. This section verifies that the device is capable of enforcing the security policy effectively.

False Positive Testing

Any signature that blocks non-malicious traffic during false positive testing was disabled for security testing.

NSS Labs Exploit Library

NSS’ security effectiveness testing leverages the deep expertise of our engineers who utilize multiple commercial, open-source, and proprietary tools, including NSS’ live stack test environment as appropriate. With 1,784 exploits, this is the industry’s most comprehensive test to date. Most notably, all of the exploits and payloads in this test were validated such that:

- A reverse shell was returned
- A bind shell was opened on the target, allowing the attacker to execute arbitrary commands
- Arbitrary code was executed
- A malicious payload was installed
- A system was rendered unresponsive
- Etc.

Product	Total Number of Attacks Run	Total Number of Attacks Blocked	Block Percentage
Palo Alto Networks PA-5220 PAN-OS 8.1.6-h2	1,784	1,783	99.94%

Figure 1 – NSS Labs Exploit Library Attacks Blocked (%)

Coverage by Attack Vector

Because a failure to block attacks could result in significant compromise and could severely impact critical business systems, NGFWs should be evaluated against a broad set of exploits. Exploits can be categorized as either *attacker-initiated* or *target-initiated*. Attacker-initiated exploits are threats executed remotely against a vulnerable application and/or operating system by an individual, while target-initiated exploits are initiated by the vulnerable target. Target-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.

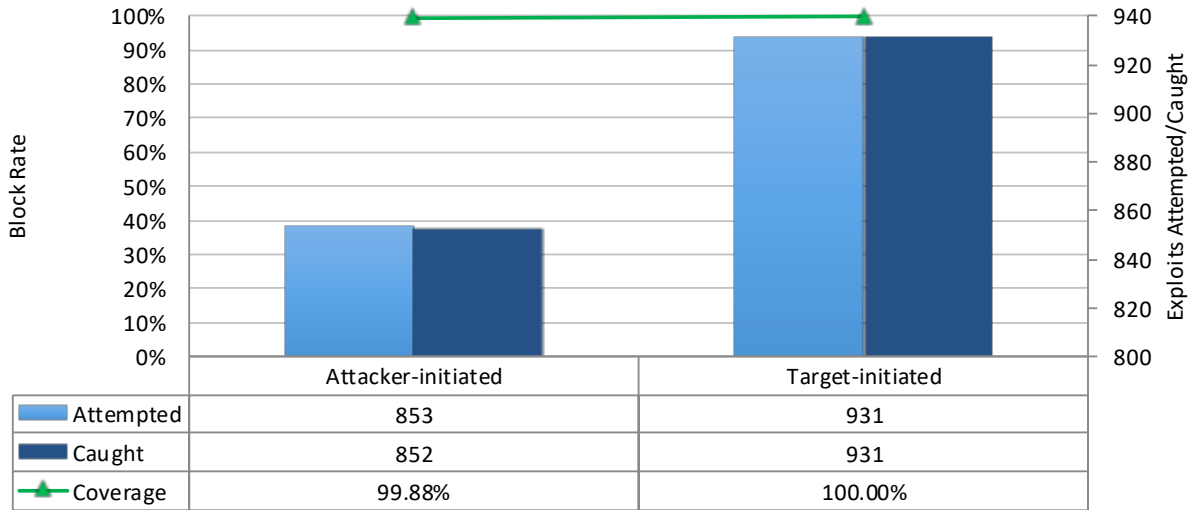


Figure 2 – Coverage by Attack Vector

Coverage by Impact Type

The most serious exploits are those that result in a remote system compromise, providing the attacker with the ability to execute arbitrary system-level commands. Most exploits in this class are “weaponized” and offer the attacker a fully interactive remote shell on the target client or server. Slightly less serious are attacks that result in individual service compromise but not arbitrary system-level command execution, although this distinction is becoming less relevant in the modern threat landscape. Finally, there are attacks that result in a system- or service-level fault that crashes the targeted service or application and requires administrative action to restart the service or reboot the system. Clients can contact NSS for more information about these tests.

Coverage by Date

Figure 3 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. NSS reports exploits by individual years for the past ten years.

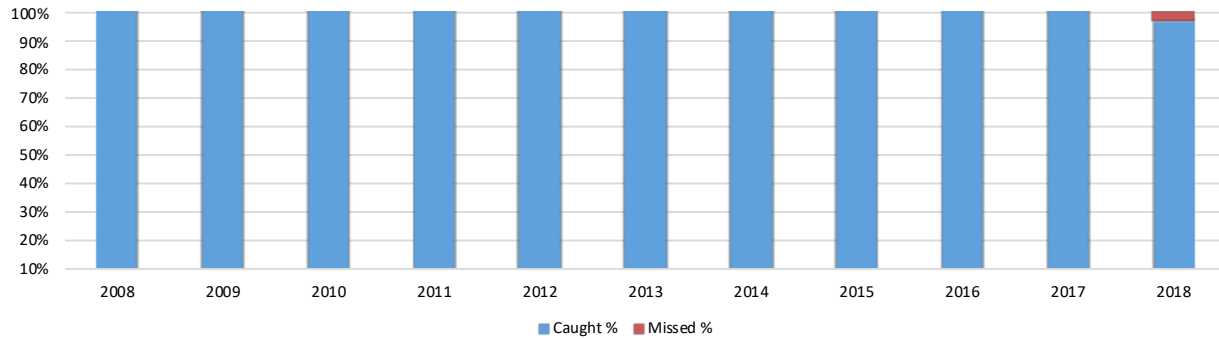


Figure 3 – Product Coverage by Date

Coverage by Target Vendor

Exploits within the *NSS Labs Exploit Library* target a wide range of protocols and applications. Figure 4 depicts the coverage offered by the PA-5220 for five of the top vendors targeted in this test. More than 70 vendors are represented in the test. Clients can contact NSS for more information.

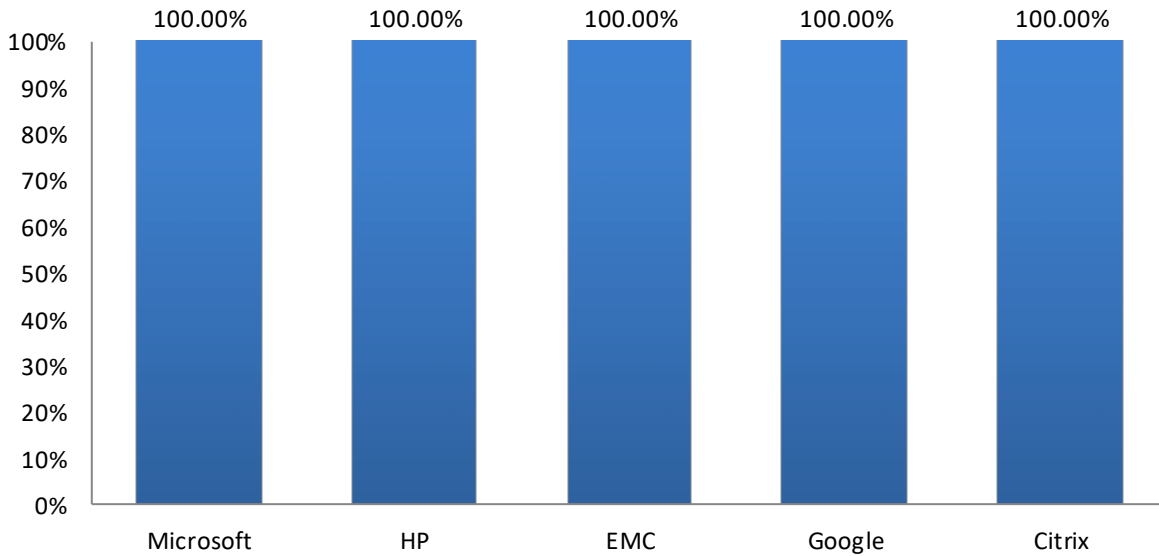


Figure 4 – Product Coverage by Target Vendor

Script Obfuscation

Figure 5 depicts how well the product protected against obfuscated scripting attacks. For additional details, please see the section on Resistance to Evasion Techniques and Appendix A: Product Scorecard. In this round of testing, no product was found capable of fully protecting against script obfuscation. Enterprises must ensure they have in place security products that can protect against these attacks.

Product	Total Number of Attacks Run	Total Number of Attacks Blocked	Block Percentage
Palo Alto Networks PA-5220 PAN-OS 8.1.6-h2	114	68	59.65%

Figure 5 – Script Obfuscation Attacks Blocked (%)

Live Exploits

This test used NSS' continuous live testing capabilities to determine how effective products are at blocking exploits that are being used, or that have been used, in active attack campaigns.³

Protection from web-based exploits targeting client applications, also known as “drive-by” downloads, can be effectively measured in NSS' unique live test harness through a series of procedures that measure the stages of protection.

Unlike traditional malware that is downloaded and installed, “drive-by” attacks first exploit a vulnerable application then silently download and install malware. For more information, see the Comparative Report on Security.

Product	Block Percentage
Palo Alto Networks PA-5220 PAN-OS 8.1.6-h2	100%

Figure 6 – Live Attacks Blocked (%)

Resistance to Evasion Techniques

Evasion techniques are a means of disguising and modifying attacks at the point of delivery to avoid detection and blocking by security products. Failure of a security device to correctly identify a specific type of evasion potentially allows an attacker to use an entire class of exploits for which the device is assumed to have protection. This often renders the device virtually useless. Many of the techniques used in this test have been widely known for years and should be considered minimum requirements for the NGFW product category.

Providing exploit protection results without fully factoring in evasions can be misleading. The more classes of evasion that are missed (such as HTTP evasions, IP packet fragmentation, TCP stream segmentation, RPC fragmentation, URL obfuscation, HTML obfuscation, resiliency, and FTP evasion), the less effective the device. For

³ See the NSS Continuous Security Validation Platform for more details.

example, it is better to miss all techniques in one evasion category, such as FTP evasion, than one technique in each category, which would result in a broader attack surface.

Furthermore, evasions operating at the lower layers of the network stack (IP packet fragmentation or stream segmentation) have a greater impact on security effectiveness than those operating at the upper layers (HTTP or FTP obfuscation). Lower-level evasions will potentially impact a wider number of exploits; missing TCP segmentation, for example, is a much more serious issue than missing FTP obfuscation.

The resiliency of a system can be defined as its ability to absorb an attack and reorganize around a threat. When an attacker is presented with a vulnerability, the attacker can select one or more paths to trigger the vulnerability. NSS introduced various previously unseen variations of exploits to exploit the vulnerability and measure the device's effectiveness against them. A resilient device can detect and prevent against different variations of an exploit. For more, see the Evasions Test Methodology v1.1 at www.nsslabs.com.

Figure 7 provides the results of the evasion tests for the PA-5220.

Test Procedure	Result
HTTP Evasions	PASS
HTTP Evasions over SSL/TLS ⁴	PASS
IPv4 Packet Fragmentation/TCP Segmentation	PASS
IPv6 Packet Fragmentation/TCP Segmentation	PASS
HTML Evasions ⁵	PASS
Resiliency ⁶	<i>See footnote</i>
Attacks on Nonstandard Ports	PASS
Combination of Evasions	PASS
RPC Fragmentation	PASS
URL Obfuscation	PASS
FTP/Telnet Evasion	PASS

Figure 7 – Resistance to Evasion Results

⁴ The evasion techniques used in the HTTPS test are the same techniques used in the HTTP test. The device was expected to decrypt, inspect, and then re-encrypt traffic.

⁵ Script obfuscations are included in the exploit block rate calculations. For details, please see Appendix A: Product Scorecard.

⁶ The results of resiliency testing are included in the exploit block rate calculations.

Performance

There is frequently a trade-off between security effectiveness and performance. Because of this trade-off, it is important to judge a product’s security effectiveness within the context of its performance and vice versa. This ensures that new security protections do not adversely impact performance and that security shortcuts are not taken to maintain or improve performance.

Raw Packet Processing Performance (UDP Throughput)

This test used UDP packets of varying sizes generated by test equipment. A constant stream of the appropriate packet size along with variable source and destination IP addresses was transmitted bidirectionally through each port pair of the device.

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 real-world network condition. The aim of the test was to determine the raw packet processing capability of each inline port pair of the device as well as the device’s effectiveness at forwarding packets quickly in order to provide the highest level of network performance with the least amount of latency.

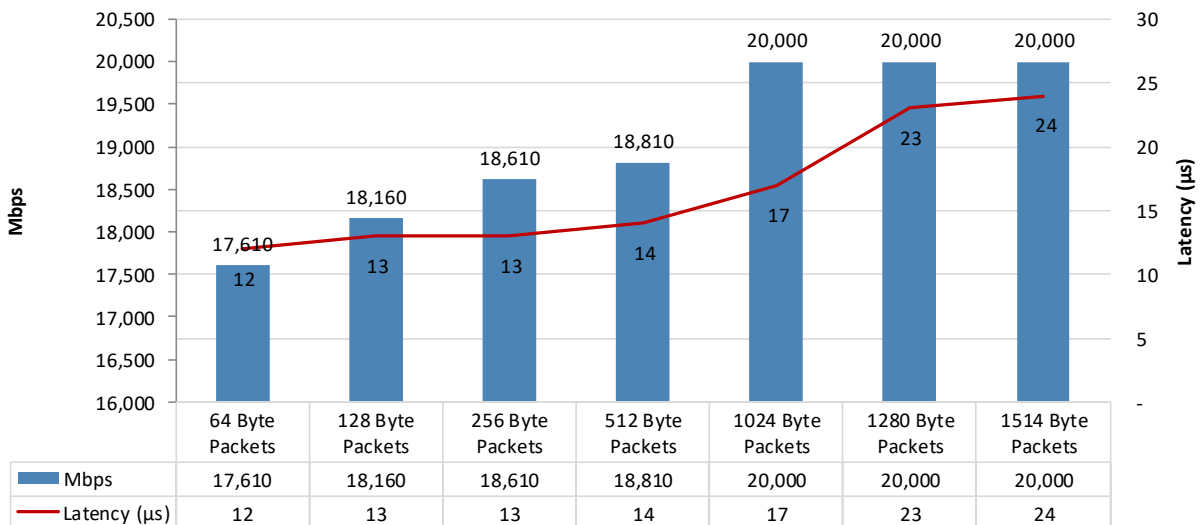


Figure 8 – Raw Packet Processing Performance (UDP Traffic)

Raw Packet Processing Performance (UDP Latency)

NGFWs that introduce high levels of latency lead to unacceptable response times for users, especially where multiple security devices are placed in the data path. Figure 9 depicts UDP latency (in microseconds) as recorded during the UDP throughput tests at 95% of maximum load.

Latency – UDP	Microseconds
64-Byte Packets	12
128-Byte Packets	13
256-Byte Packets	13
512-Byte Packets	14
1024-Byte Packets	17
1280-Byte Packets	23
1514-Byte Packets	24

Figure 9 – UDP Latency in Microseconds

Maximum Capacity

The use of traffic generation appliances allows NSS engineers to create “real-world” traffic at multi-Gigabit speeds as a background load for the tests. The aim of these tests was 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. These tests provide an excellent representation of a live network at various connection/transaction rates.

Note that in all tests, the following critical “breaking points”—where the final measurements are taken—were used:

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

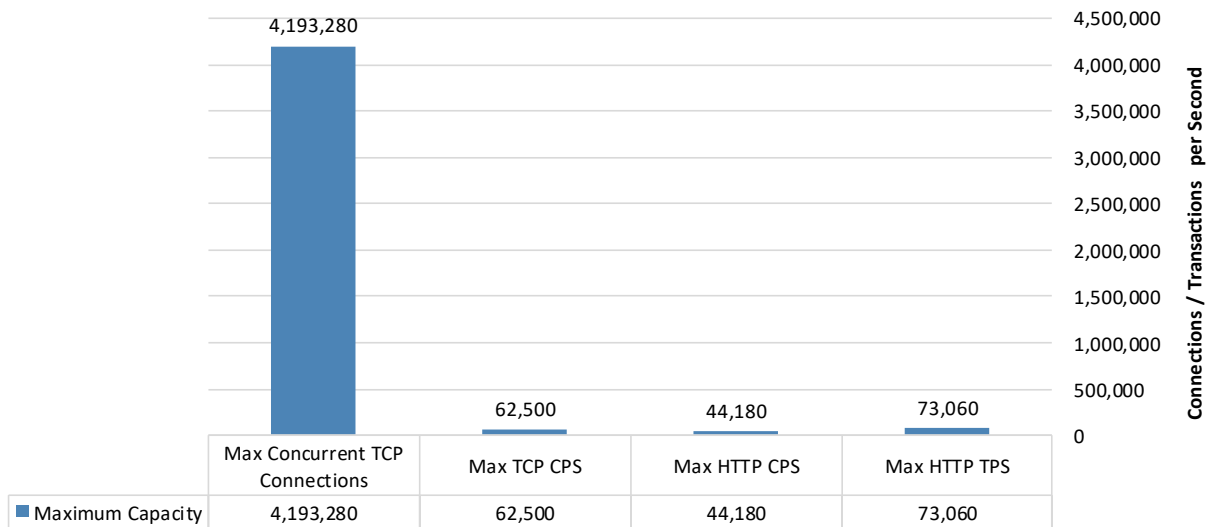


Figure 10 – Concurrency and Connection Rates

HTTP Capacity

The aim of the HTTP capacity tests was to stress the HTTP detection engine and determine how it copes with network loads of varying average packet size and varying connections per second. By creating multiple tests using genuine session-based traffic with varying session lengths, the device was forced to track valid HTTP sessions, thus ensuring a higher workload than for simple packet-based background traffic.

Each transaction consisted of a single HTTP GET request. All packets contained valid payload (a mix of binary and ASCII objects) and address data. These tests provide an excellent representation of a live network (albeit one biased toward HTTP traffic) at various network loads.

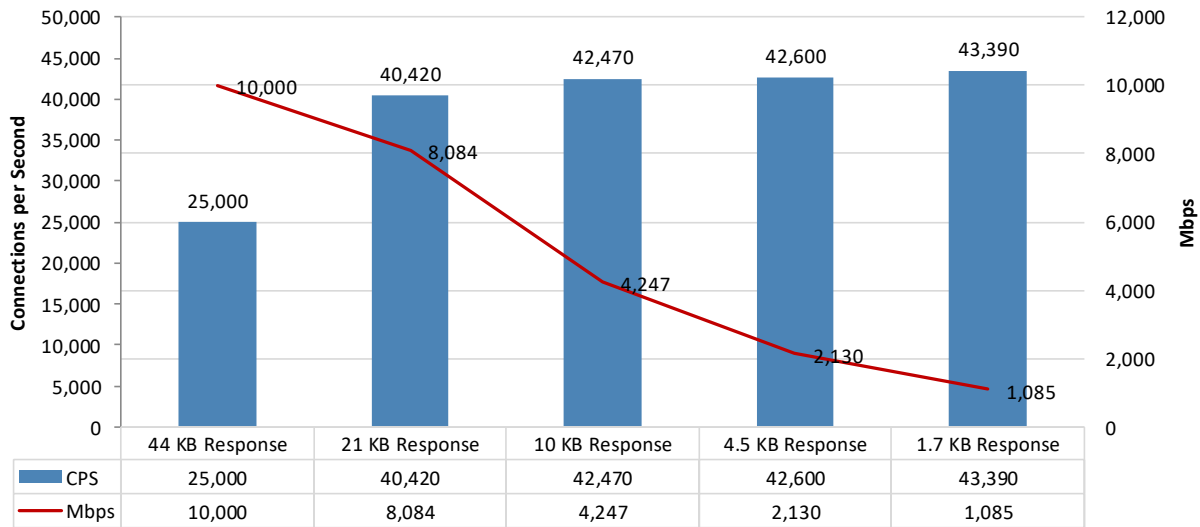


Figure 11 – HTTP Capacity

Application Average Response Time – HTTP

Application Average Response Time – HTTP (at 95% Maximum Load)	Milliseconds
2,500 Connections per Second – 44 KB Response	1.55
5,000 Connections per Second – 21 KB Response	1.18
10,000 Connections per Second – 10 KB Response	1.15
20,000 Connections per Second – 4.5 KB Response	1.11
40,000 Connections per Second – 1.7 KB Response	6.40

Figure 12 – Average Application Response Time (Milliseconds)

HTTP Capacity with HTTP Persistent Connections

These tests used HTTP persistent connections with each TCP connection containing 10 HTTP GETs and associated responses. All packets contained valid payload (a mix of binary and ASCII objects) and address data. These tests provide an excellent representation of a live network at various network loads. The stated response size is the total of all HTTP responses within a single TCP session.

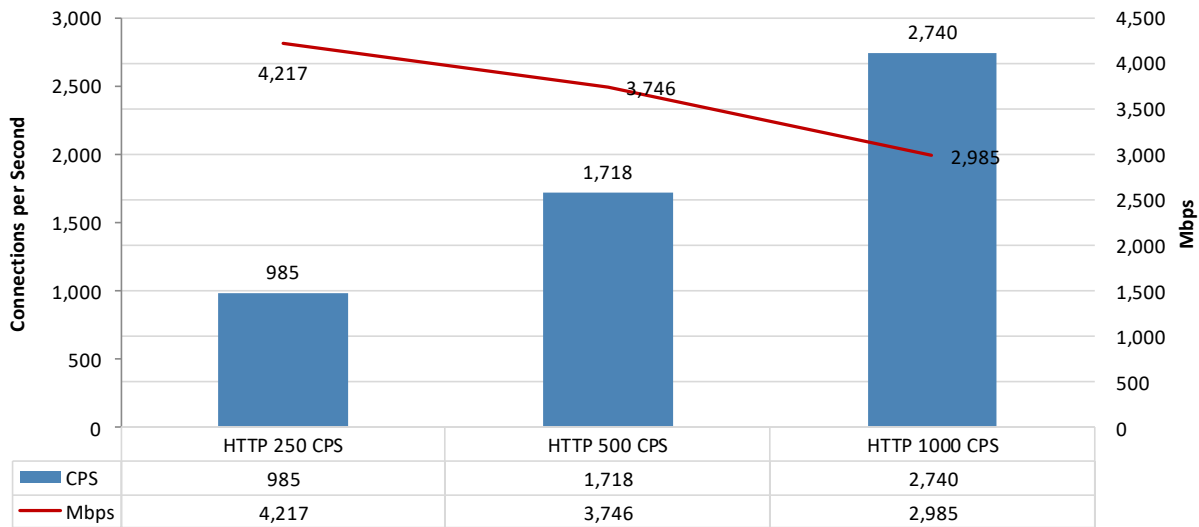


Figure 13 – HTTP Capacity with HTTP Persistent Connections

HTTP Capacity with Bidirectional HTTP Persistent Connections

These tests used bidirectional HTTP persistent connections, with each TCP connection containing 10 HTTP GETs and associated responses. All packets contained valid payload (a mix of binary and ASCII objects) and address data. These tests provide an excellent representation of a live network at various network loads. The stated response size is the total of all HTTP responses within a single TCP session.

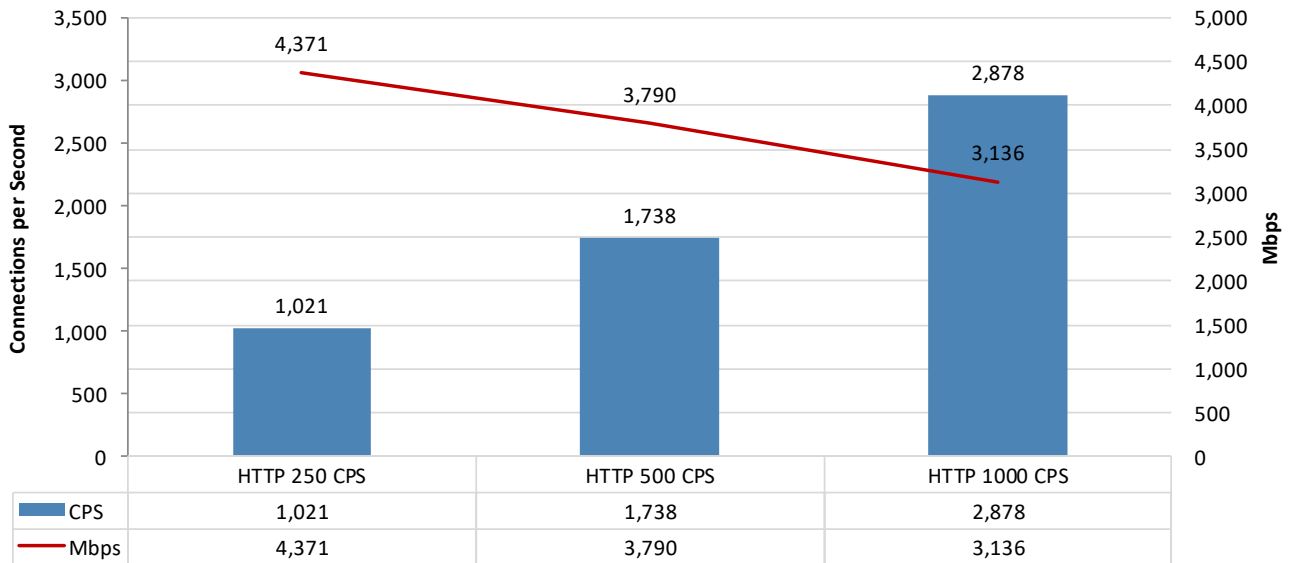


Figure 14 – HTTP Capacity with Bidirectional HTTP Persistent Connections

Single Application Flows

These tests measured the performance of the device with single application flows. For details about single application flow testing, see the NSS Labs Next Generation Firewall Test Methodology v9.0, available at www.nsslabs.com.

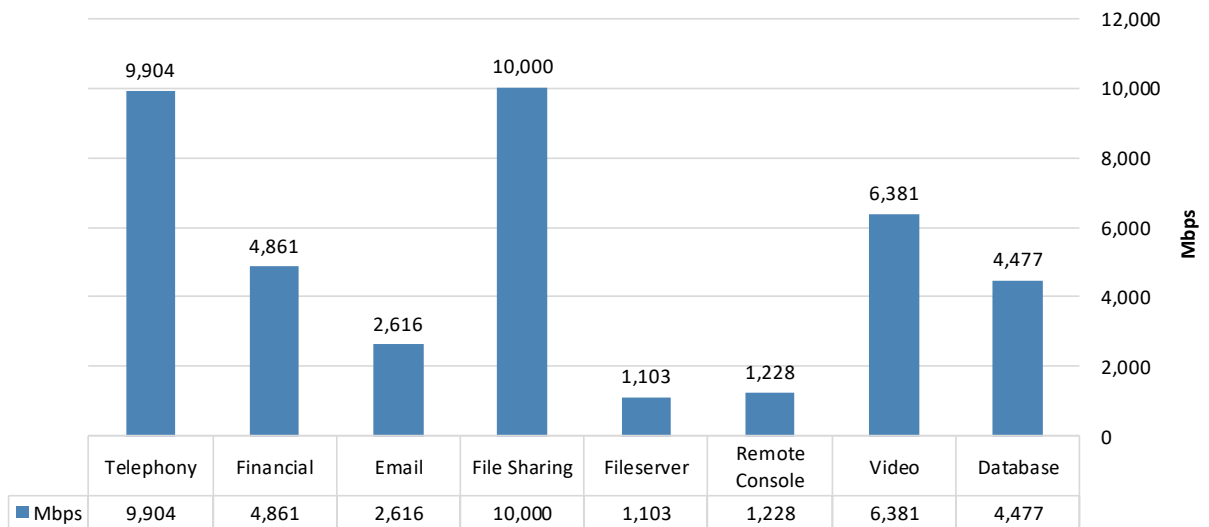


Figure 15 – Single Application Flows

HTTP NSS-Tested Throughput

HTTP NSS-Tested Throughput (Mbps) is calculated as a weighted average of the traffic that NSS expects an NGFW to experience in an enterprise environment where no HTTPS traffic is present. For more details, please see Appendix A: Product Scorecard.

Product	HTTP NSS-Tested Throughput (Mbps)
Palo Alto Networks PA-5220 PAN-OS 8.1.6-h2	7,737

Figure 16 – HTTP NSS-Tested Throughput (Mbps)

SSL/TLS

Use of the Secure Sockets Layer (SSL) protocol and its newer iteration, Transport Layer Security (TLS), has risen in accordance with the increasing need for privacy online. Modern cybercampaigns frequently focus on attacking users through the most common web protocols and applications. NSS continues to receive inquiries from enterprise customers during their assessments of vendors that provide SSL/TLS decryption and protection technologies. To this end, NSS tests the capabilities and performance of devices providing SSL/TLS visibility.

SSL/TLS Functionality

Decryption Validation

To confirm that the device under test was correctly decrypting and (if applicable) inspecting SSL/TLS traffic, a validation test was performed prior to the functional or performance testing. The device was expected to cover all test cases in this methodology with a single configuration.

For devices providing both decryption and inspection, this test consisted of a known exploit embedded in encrypted traffic being passed through the device. NSS has an extensive library of well-known malicious files and exploits suitable for this purpose. Devices in this category were expected to decrypt the stream, detect the exploit, and block or alert as appropriate. The purpose of this test was not to evaluate the security effectiveness of the device but rather to validate that the device was decrypting and inspecting traffic.

For devices providing decryption without an onboard inspection engine (i.e., SSL/TLS offload devices), the same test was performed. However, instead of requiring the device to block or alert on the encapsulated payload, the evaluation methodology includes manual analysis via differential comparison of the traffic pre- and post-proxy.

Cipher Selection

To determine the most commonly employed cipher suites for inclusion in testing, ciphers were selected from the 12/31/2017 results of the Alexa Top 1 Million Analysis.⁷ The top 30 ciphers from this data were selected for use in functional capability testing and the top four (representing more than 90% of the distribution) were selected for use in performance testing.

Cipher Support

The device was expected to be capable of negotiating a wide range of commonly used SSL/TLS ciphers in order to increase the security visibility of potential threats encapsulated in real-world SSL/TLS traffic. This test covered the top 30 cipher suites. Unless otherwise specified, the functional tests used the most common key sizes for RSA (2,048 bit) and ECDSA (256 bit).

⁷ Alexa Top 1 Million Analysis performed on 12/31/2017 by Scott Helme; methodology in SSL/TLS Performance Test Methodology v1.3 020218 Appendix A: Cipher Selection Details.

Top 30 Cipher Suites from the Alexa Top 1 Million⁸

- TLS ECDHE RSA WITH AES 256 GCM SHA384
- TLS ECDHE RSA WITH AES 128 GCM SHA256
- TLS ECDHE ECDSA WITH AES 128 GCM SHA256
- TLS ECDHE RSA WITH AES 256 CBC SHA384
- TLS DHE RSA WITH AES 256 GCM SHA384
- TLS ECDHE RSA WITH AES 256 CBC SHA
- TLS DHE RSA WITH AES 256 CBC SHA
- TLS RSA WITH AES 256 CBC SHA
- TLS ECDHE RSA WITH AES 128 CBC SHA
- TLS ECDHE ECDSA WITH AES 256 GCM SHA384
- TLS RSA WITH RC4 128 MD5
- TLS ECDHE RSA WITH RC4 128 SHA
- TLS DHE RSA WITH AES 128 CBC SHA
- TLS DHE RSA WITH AES 128 GCM SHA256
- TLS RSA WITH 3DES EDE CBC SHA
- TLS DHE RSA WITH AES 256 CBC SHA256
- TLS RSA WITH AES 128 CBC SHA
- TLS RSA WITH AES 256 CBC SHA256
- TLS RSA WITH AES 256 GCM SHA384
- TLS ECDHE RSA WITH AES 128 CBC SHA256
- TLS RSA WITH AES 128 CBC SHA256
- TLS RSA WITH RC4 128 SHA
- TLS RSA WITH AES 128 GCM SHA256
- TLS DHE RSA WITH CAMELLIA 256 CBC SHA
- TLS DHE RSA WITH SEED CBC SHA
- TLS RSA WITH SEED CBC SHA
- TLS ECDHE RSA WITH 3DES EDE CBC SHA
- TLS RSA WITH CAMELLIA 256 CBC SHA
- TLS DHE RSA WITH 3DES EDE CBC SHA
- TLS DHE RSA WITH AES 128 CBC SHA256

Support for Emergent Ciphers

Support for the following emergent ciphers was also tested:

- TLS ECDHE ECDSA WITH CHACHA20 POLY1305 SHA256
- TLS ECDHE RSA WITH CHACHA20 POLY1305 SHA256

Support for x25519 Elliptic Curve

Support for the x25519 Elliptic Curve parameter was also tested during the Emergent Cipher test.

Prevention of Weak Ciphers

The device was expected to protect against the use of ciphers that are known to offer either weak protection or none at all, including (but not limited to):

- Null ciphers (no encryption of data provided)
- Anonymous ciphers (no authentication provided)

Decryption Bypass Exceptions

The device was expected to support the configuration of policies that permit conditional bypass of decryption in order to preserve privacy, either for regulatory or other reasons. Devices must maintain decryption capabilities as tested in the *Cipher Support* section concurrently with these conditional bypass rules; i.e., turning off all decryption on a device is not an acceptable method for meeting requirements in this section. The device was tested for decryption bypass capabilities under various conditions, including:

⁸ As of 12/31/2017

- Layer 3 information (i.e., bypass based on source or destination IP address)
- Layer 4 information (i.e., bypass based on TCP port number)
- Server Name Indication (SNI) TLS extension information
- Site category based on Common Name (CN) and/or Subject Alternative Name (SAN)

Certificate Validation

The device was expected to validate the status of all SSL/TLS certificates presented, except in cases where decryption bypass was enabled. When presented with an invalid certificate, the device had to either prevent the establishment of a connection or replicate the original invalid status in the proxied/resigned certificate presented to the client, such that the client was aware of the potential risk.

TLS Session Reuse

In order 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. The two primary methods for session reuse are session IDs and session tickets. Whereas session IDs are included in the main TLS specification, session tickets are an extension of the specification, detailed in a separate RFC. Support for both of these methods was tested under this section.

These tests assessed the scope of support for a wide range of cipher suites, including functional checks for common extensions to the TLS protocol, as well as policies to bypass the decryption process for certain subsets of traffic.

SSL/ TLS Functionality Tested	Score
Decryption validation	PASS
Top 30 cipher suite support ⁹	20/24
Emergent ciphers	0/2
Support for x25519 Elliptic Curve	FAIL
Prevention of weak ciphers	PASS
Decryption bypass policy based on Layer 3 information	PASS
Decryption bypass policy based on Layer 4 information	PASS
Decryption bypass policy based on Server Name Indication (SNI) TLS extension information	PASS
Decryption bypass policy based on Common Name (CN) and/or Subject Alternative Name (SAN)	PASS
Certificate validation	PASS
Support for session ID reuse method	PASS
Support for session ticket reuse method	PASS

Figure 17 – SSL/TLS Functionality Tested

⁹ Six of the top 30 cipher suites (RC4 and 3DES cipher suites) are considered not secure. Please contact your vendor to see if ciphers are supported.

Maximum SSL/TLS Handshakes per Second

This test was designed to determine the maximum HTTPS connection rate of the device with a one-byte response size. This type of traffic is atypical of a normal network, but the negligible payload size provides a means to measure the device’s SSL/TLS handshake performance independent of throughput performance. An increasing number of new sessions was established through the device until a maximum is reached, and each session was immediately closed on successful negotiation of the SSL/TLS handshake and transfer of the payload.

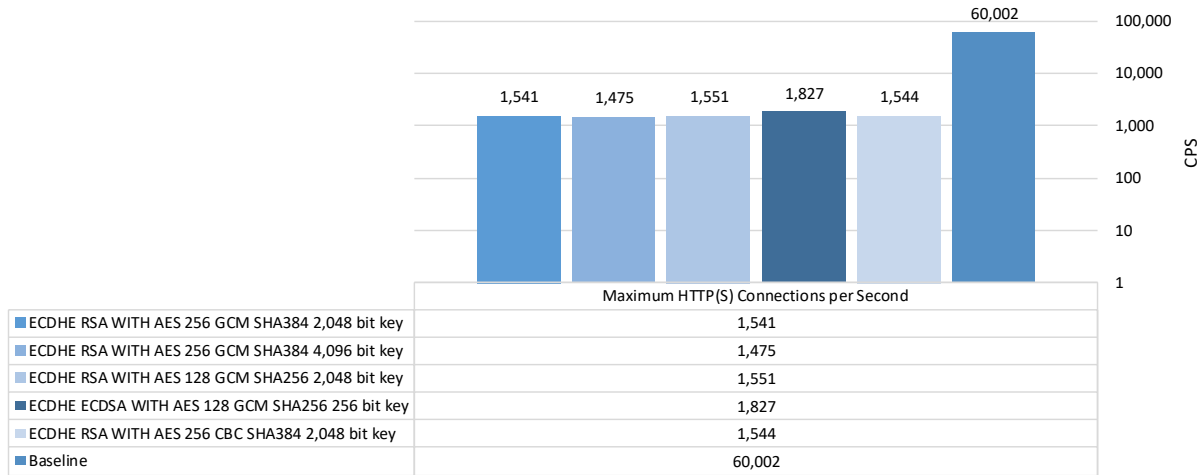


Figure 18 – Maximum HTTP(S) Connections per Second

HTTPS Throughput Capacity

The aim of these tests was to stress the HTTPS engine and determine how the device copes with network loads of varying average packet size 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 for simple packet-based background traffic. This provides a test environment that is as close to real-world conditions as it is possible to achieve in a lab environment, while ensuring accuracy and repeatability. Each transaction consisted of either a single (1) HTTP(S) GET request or ten (10) HTTP(S) GETs, 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. These tests provide a feasible representation of a live network (albeit one biased toward HTTPS traffic) at various network loads.

Figure 19 through Figure 26 depict the results of the HTTPS Throughput Capacity tests.

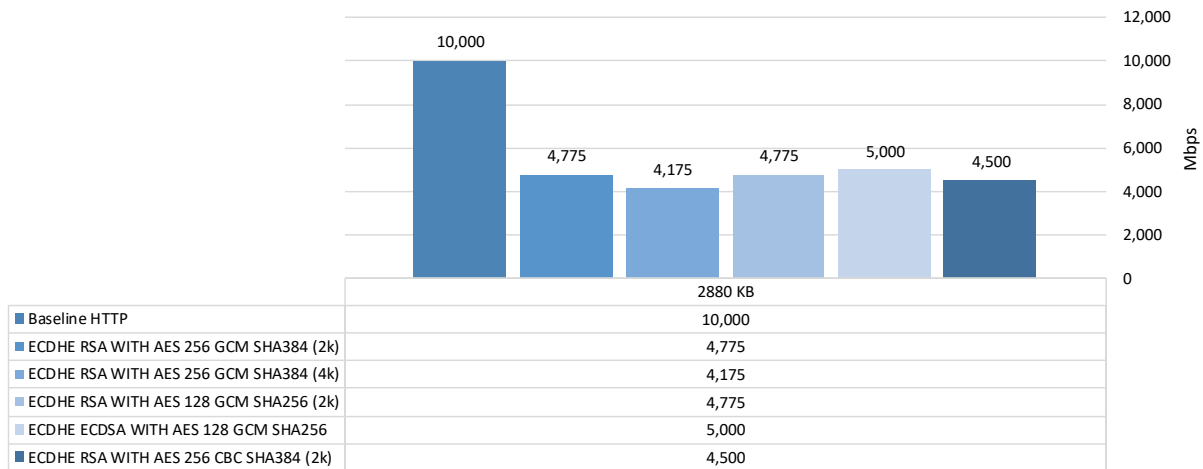


Figure 19 – HTTP Capacity (No Persistence) Single HTTP GET Request (2,880 KB)

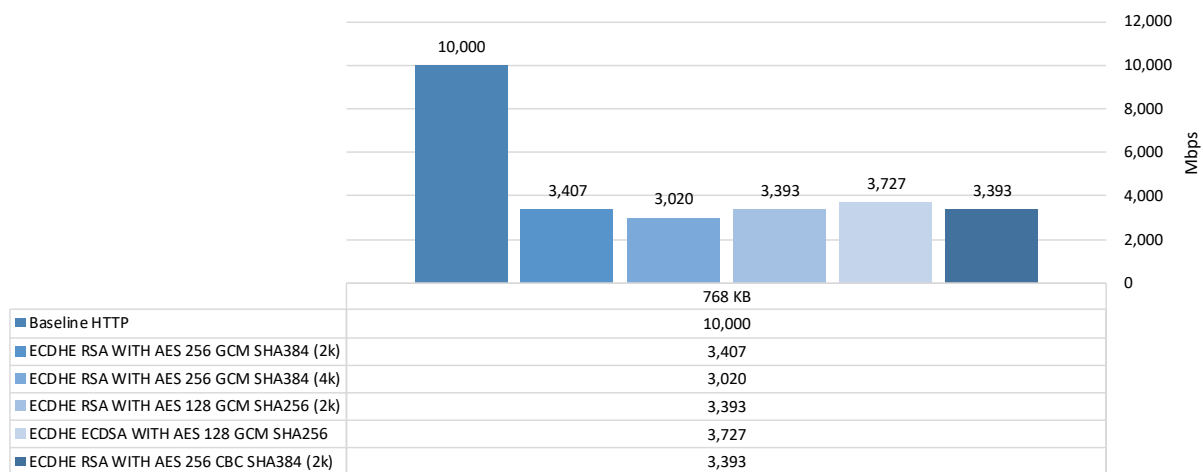


Figure 20 – HTTP Capacity (No Persistence) Single HTTP GET Request (768 KB)

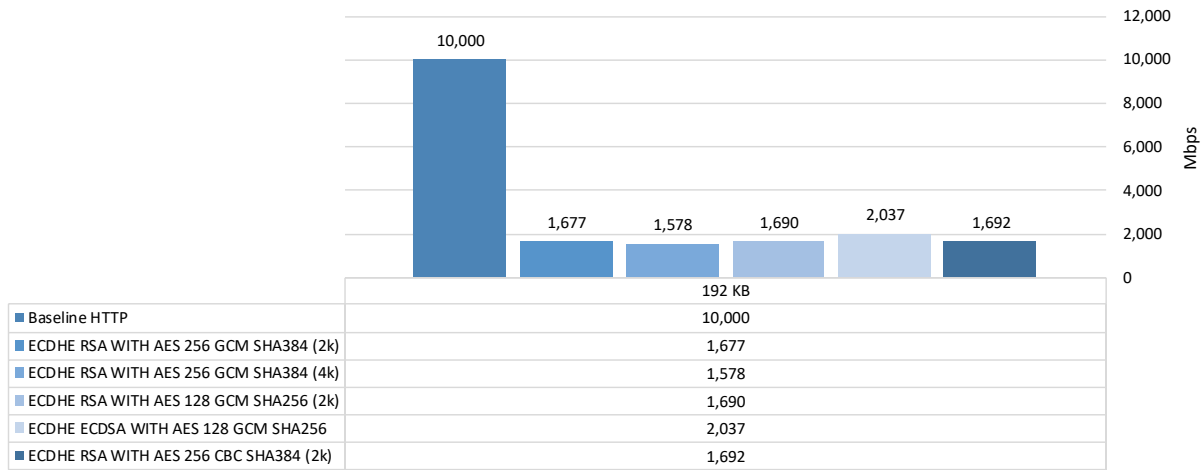


Figure 21 – HTTP Capacity (No Persistence) Single HTTP GET Request (192 KB)

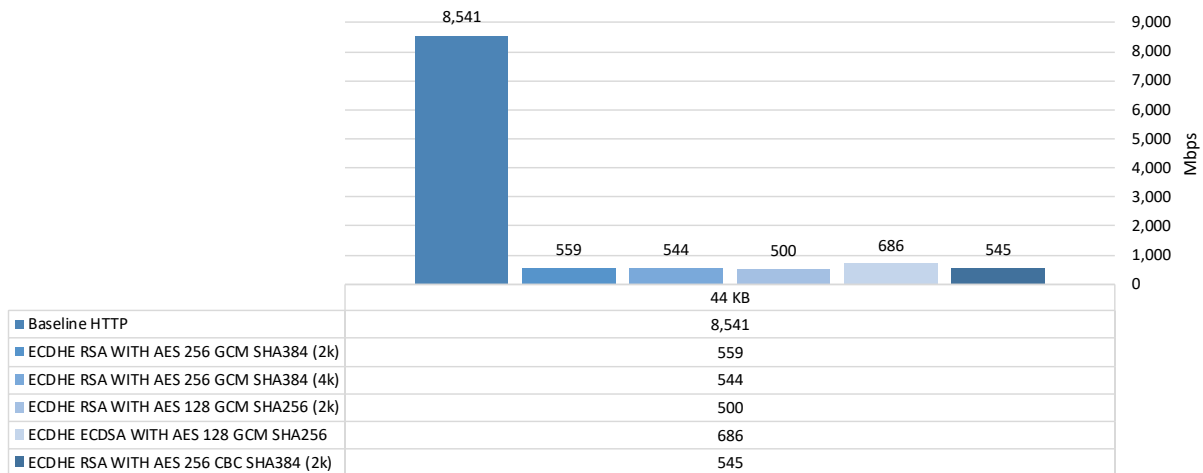


Figure 22 – HTTP Capacity (No Persistence) Single HTTP GET Request (44 KB)

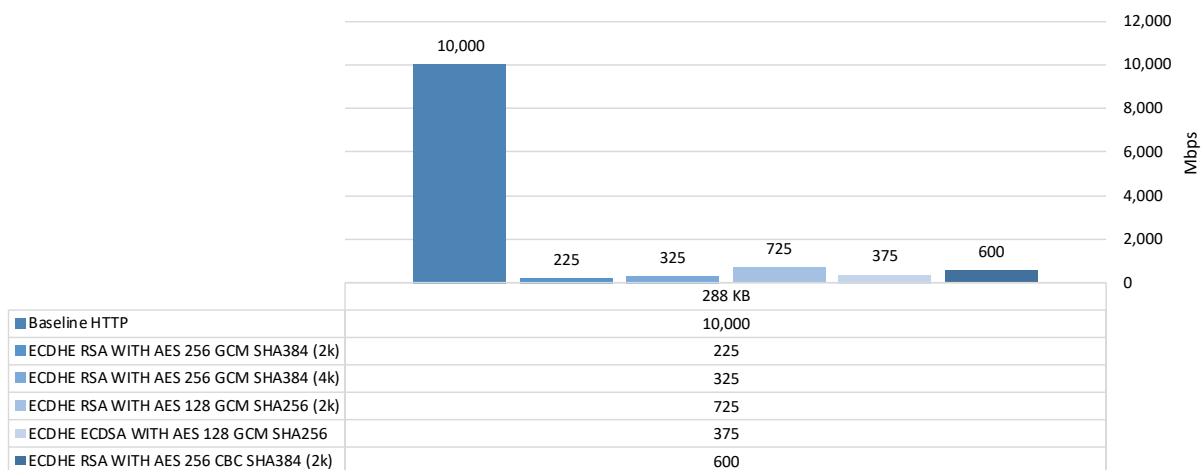


Figure 23 – HTTP Capacity with Persistent Connections with 10 HTTP GET Requests (288 KB)

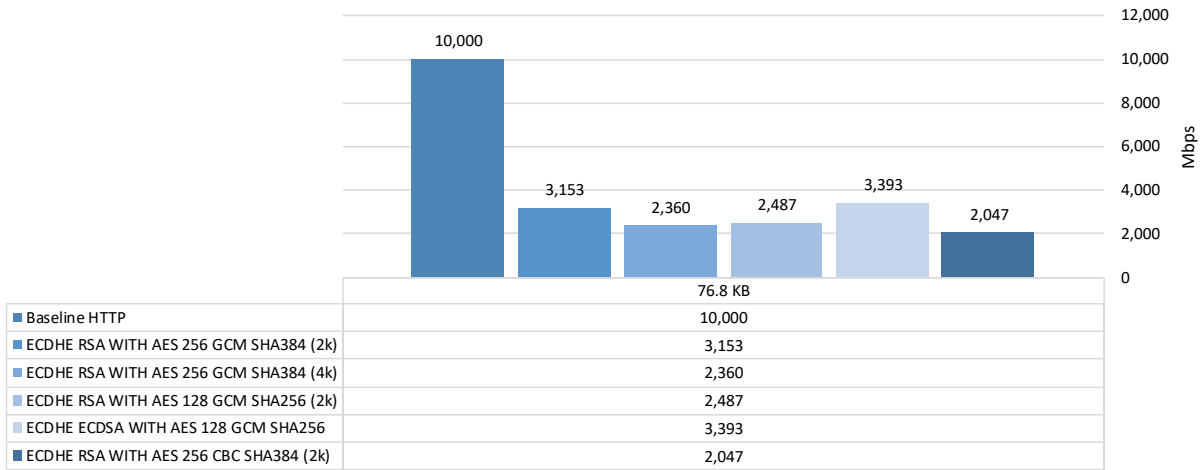


Figure 24 – HTTP Capacity with Persistent Connections with 10 HTTP GET Requests (76.8 KB)

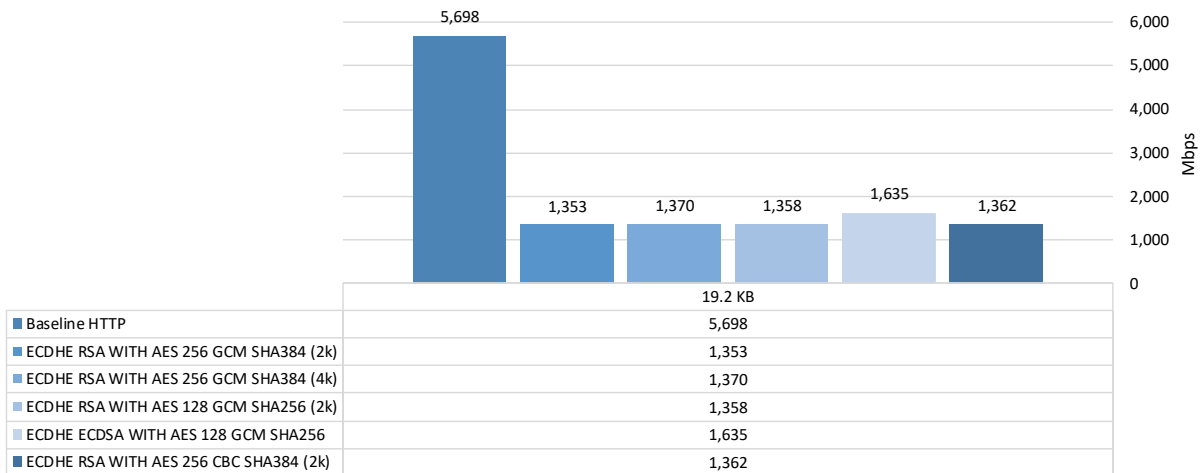


Figure 25 – HTTP Capacity with Persistent Connections with 10 HTTP GET Requests (19.2KB)

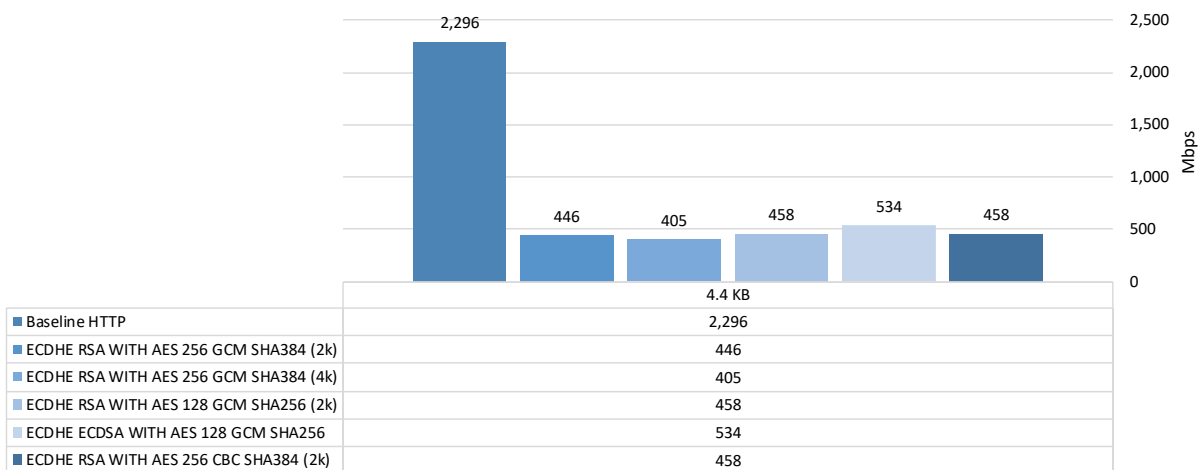


Figure 26 – HTTP Capacity with Persistent Connections with 10 HTTP GET Requests (4.4 KB)

HTTPS (SSL/TLS) NSS-Tested Throughput

HTTPS (SSL/TLS) NSS-Tested Throughput (Mbps) is calculated as a weighted average of the traffic that NSS expects an NGFW to experience in an enterprise environment where SSL traffic is present. For more details, please see Appendix A: Product Scorecard.

Product	HTTPS (SSL/TLS) NSS-Tested Throughput (Mbps)
Palo Alto Networks PA-5220 PAN-OS 8.1.6-h2	1,913

Figure 27 – HTTPS (SSL/TLS) NSS-Tested Throughput (Mbps)

Stability and Reliability

Long-term stability is particularly important for an inline device, where failure can produce a network outage. These tests verified the device’s ability to block malicious traffic while under extended load as well as sustain legitimate traffic.

The device was required to remain operational and stable throughout all these tests and to block 100% of previously known malicious attacks, raising an alert for each. If any non-allowed traffic passed successfully, caused either by the volume of traffic or by the device failing open for any reason, the device failed the test.

Stability and Reliability	Result
Blocking under Extended Attack	PASS
Passing Legitimate Traffic under Extended Attack	PASS
Behavior of the State Engine under Load	
<ul style="list-style-type: none"> Attack Detection/Blocking – Normal Load 	PASS
<ul style="list-style-type: none"> State Preservation – Normal Load 	PASS
<ul style="list-style-type: none"> Pass Legitimate Traffic – Normal Load 	PASS
<ul style="list-style-type: none"> Drop Traffic – Maximum Exceeded 	PASS
Power Fail	PASS
Backup/Restore	PASS
Persistence of Data	PASS
Stability	PASS

Figure 28 – Stability and Reliability Results

Total Cost of Ownership (TCO)

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

- **Product Purchase** – The cost of acquisition.
- **Product Maintenance** – The fees paid to the vendor, including software and hardware support, maintenance, and other updates.
- **Installation** – The time required to take the device out of the box, configure it, put it into the network, apply updates and patches, and set up desired logging and reporting.
- **Upkeep** – The time required to apply periodic updates and patches from vendors, including hardware, software, and other updates.
- **Management** – Day-to-day management tasks, including device configuration, policy updates, policy deployment, alert handling, and so on.

For the purposes of this report, capital expenditure (capex) items are included for a single device only (the cost of acquisition and installation).

Installation Hours

Figure 29 depicts the number of hours of labor required to install each device using only local device management options. The table accurately reflects the amount of time that NSS engineers, with the help of vendor engineers, needed to install and configure the device to the point where it operated successfully in the test harness, passed legitimate traffic, and blocked and detected prohibited or malicious traffic. This closely mimics a typical enterprise deployment scenario for a single device.

The installation cost is based on the time that an experienced security engineer would require to perform the installation tasks described above. This approach allows NSS to hold constant the talent cost and measure only the difference in time required for installation. Readers should substitute their own costs to obtain accurate TCO figures.

Product	Installation (Hours)
Palo Alto Networks PA-5220 PAN-OS 8.1.6-h2	8

Figure 29 – Sensor Installation Time (Hours)

Total Cost of Ownership

Calculations are based on vendor-provided pricing information. Where possible, the 24/7 maintenance and support option with 24-hour replacement is utilized, since this is the option typically selected by enterprise customers. Prices are for single device management and maintenance only; costs for central management solutions (CMS) may be extra.

Product	Year 1 Cost	Year 2 Cost	Year 3 Cost	3-Year TCO
Palo Alto Networks PA-5220 PAN-OS 8.1.6-h2	\$48,900	\$10,800	\$10,800	\$70,500

Figure 30 –3-Year TCO (US\$)

- **Year 1 Cost** is calculated by adding installation costs (US\$75 per hour fully loaded labor x installation time) + 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.

For additional TCO analysis, including for the CMS, refer to the TCO Comparative Report.

HTTP/1.1 response with transfer-encoding header declaring chunking with lots of whitespace ('Transfer-Encoding:' followed by 8000 spaces (hex '20' * 8000) followed by 'chunked' followed by CRLF (hex '0d 0a'); served chunked	PASS
HTTP/1.0 response declaring chunking; served without chunking	PASS
HTTP/1.0 response declaring chunking with invalid content-length header; served without chunking	PASS
HTTP/1.1 response with "\tTransfer-Encoding: chunked"; served chunked	PASS
HTTP/1.1 response with "\tTransfer-Encoding: chonked" after custom header line with "chunked" as value; served without chunking	PASS
HTTP/1.1 response with header with no field name and colon+junk string; followed by '\tTransfer-Encoding: chunked' header; followed by custom header; served chunked	PASS
HTTP/1.1 response with "\r\rTransfer-Encoding: chunked"; served chunked	PASS
HTTP/1.1 response with using single "\n"s instead of "\r\n"s; chunked	PASS
HTTP/1.1 response with \r\n\r\n before first header; chunked	PASS
HTTP/1.1 response with "SIP/2.0 200 OK\r\n" before status header; chunked	PASS
HTTP/1.1 response with space+junk string followed by \r\n before first header; chunked	PASS
HTTP/1.1 response with junk string before status header; chunked	PASS
HTTP/1.1 response with header end \n\004\n; chunked	PASS
HTTP/1.1 response with header end \r\n\010\r\n\r\n; chunked	PASS
HTTP/1.1 response with header end \n\r\r\n; chunked	PASS
HTTP/1.1 response with header end \n\006\011\n; chunked	PASS
HTTP/1.1 response with header end \n\033\n\003\n; chunked	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 status code 202; with message-body; chunked	PASS
HTTP/1.1 response with status code 429; with message-body; chunked	PASS
HTTP/1.1 response with status code 300; with message-body; chunked	PASS
HTTP/1.1 response with status code 306; with message-body; chunked	PASS
HTTP/1.1 response with status code 414; with message-body; chunked	PASS
HTTP/1.1 chunked response with no status indicated	PASS
No status line; chunking indicated; served unchunked	PASS
HTTP/1.1 response with invalid content-length header size declaration followed by space and null (hex '20 00')	PASS
Version HTTP/2.0 declared; served chunked	PASS
Version HTTP/0001.1 declared; served chunked	PASS
Version HTTP/6.-66 declared; served chunked	PASS
Version HTTP/7.7 declared; served chunked	PASS
Double Transfer-Encoding: first empty; last chunked. Served with invalid content-length; not chunked.	PASS
Relevant headers padded by preceding with hundreds of random custom headers; chunked	PASS
HTTP/1.1 response with "Transfer-Encoding: chunked header; not 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'); chunk with some data in chunk-extension field	PASS
HTTP/1.0 response declaring chunking; chunk with some data in chunk-extension field	PASS
HTTP/1.0 response declaring chunking with invalid content-length header; chunk with some data in chunk-extension field	PASS
HTTP/1.1 response with "\tTransfer-Encoding: chonked" after custom header line with "chunked" as value; chunk with some data in chunk-extension field	PASS

HTTP/1.1 response with using single "\n"s instead of "\r\n"s; chunked	PASS
HTTP/1.1 response with \r\n\r\n before first header; chunked	PASS
HTTP/1.1 response with "SIP/2.0 200 OK\r\n" before status header; chunked	PASS
HTTP/1.1 response with space+junk string followed by \r\n before first header; chunked	PASS
HTTP/1.1 response with junk string before status header; chunked	PASS
HTTP/1.1 response with header end \n\014\n; chunked	PASS
HTTP/1.1 response with header end \r\n\016\r\n\r\n; chunked	PASS
HTTP/1.1 response with header end \n\r\r\n; chunked	PASS
HTTP/1.1 response with header end \n\017\018\n; chunked	PASS
HTTP/1.1 response with header end \n\030\n\019\n; chunked	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 status code -203.030; with message-body; chunked	PASS
HTTP/1.1 response with status code 402; with message-body; chunked	PASS
HTTP/1.1 response with status code 403; with message-body; chunked	PASS
HTTP/1.1 response with status code 406; with message-body; chunked	PASS
HTTP/1.1 response with status code 505; with message-body; chunked	PASS
HTTP/1.1 chunked response with no status indicated	PASS
No status line; chunking indicated; served unchunked	PASS
HTTP/1.1 response with invalid content-length header size declaration followed by space and null (hex '20 00')	PASS
Version HTTP/1.01 declared; served chunked	PASS
Version HTTP/01.1 declared; served chunked	PASS
Version HTTP/2.B declared; served chunked	PASS
Version HTTP/9.-1 declared; served chunked	PASS
Double Transfer-Encoding: first empty; last chunked. Served with invalid content-length; not chunked.	PASS
Relevant headers padded by preceding with hundreds of random custom headers	PASS
HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes followed by backspace (hex '08'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes followed by end of text (hex '03'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes followed by escape (hex '1b'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes followed by null (hex '00'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a zero (hex '30'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'); compressed with deflate	PASS
HTTP/1.1 chunked response with chunk sizes followed by backspace (hex '08'); compressed with deflate	PASS
HTTP/1.1 chunked response with chunk sizes followed by end of text (hex '03'); compressed with deflate	PASS
HTTP/1.1 chunked response with chunk sizes followed by escape (hex '1b'); compressed with deflate	PASS
HTTP/1.1 chunked response with chunk sizes followed by null (hex '00'); compressed with deflate	PASS
HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a zero (hex '30'); compressed with deflate	PASS

IPv4 Packet Fragmentation/TCP Segmentation	
Small IP fragments; overlapping duplicate fragments with garbage payloads	PASS
Small IP fragments in reverse order	PASS
Small IP fragments in random order	PASS
Small IP fragments; delay first fragment	PASS
Small IP fragments in reverse order; delay last fragment	PASS
Small IP fragments; interleave chaff after (invalid IP options)	PASS
Small IP fragments in random order; interleave chaff sandwich (invalid IP options)	PASS
Small IP fragments in random order; interleave chaff sandwich (invalid IP options); delay random fragment	PASS
Small IP fragments; interleave chaff before (invalid IP options); DSCP value 16	PASS
Small IP fragments in random order; interleave chaff after (invalid IP options); delay random fragment; DSCP value 34	PASS
IPv4 fragmentation with an overlapping atomic fragment with good data inserted in-between the fragments with junk data	PASS
IPv4 fragmentation with an overlapping atomic fragment with junk data inserted in-between the fragments with good data	PASS
IPv4 fragmentation with an overlapping atomic fragment with good data inserted in-between the fragments with junk data; chunked	PASS
IPv4 fragmentation with an overlapping atomic fragment with junk data inserted in-between the fragments with good data; chunked	PASS
IPv4 fragmentation with an overlapping atomic fragment with good data inserted in-between the fragments with junk data; compressed with gzip	PASS
IPv4 fragmentation with an overlapping atomic fragment with junk data inserted in-between the fragments with good data; compressed with gzip	PASS
IPv4 fragmentation with an overlapping atomic fragment with good data inserted in-between the fragments with junk data; chunked; compressed with deflate	PASS
IPv4 fragmentation with an overlapping atomic fragment with junk data inserted in-between the fragments with good data; chunked; compressed with deflate	PASS
Small TCP segments; overlapping duplicate segments with garbage payloads	PASS
Small TCP segments in reverse order	PASS
Small TCP segments in random order	PASS
Small TCP segments; delay first segment	PASS
Small TCP segments in reverse order; delay last segment	PASS
Small TCP segments; interleave chaff after (invalid TCP checksums); delay first segment	PASS
Small TCP segments in random order; interleave chaff before (invalid TCP checksums); delay random segment	PASS
Small TCP segments in random order; interleave chaff sandwich (out-of-window sequence numbers); TCP MSS option	PASS
Small TCP segments in random order; interleave chaff after (requests to resynch sequence numbers mid-stream); TCP window scale option	PASS
Small TCP segments in random order; interleave chaff sandwich (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment	PASS
Small overlapping TCP segments	PASS
Small overlapping TCP segments; method 2	PASS
Small overlapping TCP segments; method 3	PASS
Small overlapping TCP segments; chunked	PASS
Small overlapping TCP segments; method 2; chunked	PASS
Small overlapping TCP segments; method 3; chunked	PASS
Small overlapping TCP segments; chunked; compressed with gzip	PASS
Small overlapping TCP segments; method 2; chunked; compressed with gzip	PASS

Small overlapping TCP segments; method 3; chunked; compressed with gzip	PASS
Small overlapping TCP segments; chunked; compressed with deflate	PASS
Small overlapping TCP segments; method 2; chunked; compressed with deflate	PASS
Small overlapping TCP segments; method 3; chunked; compressed with deflate	PASS
Small TCP segments; small IP fragments	PASS
Small TCP segments; small IP fragments in reverse order	PASS
Small TCP segments in random order; small IP fragments	PASS
Small TCP segments; small IP fragments in random order	PASS
Small TCP segments in random order; small IP fragments in reverse order	PASS
Small TCP segments in random order; interleave chaff sandwich (invalid TCP checksums); small IP fragments in reverse order; interleave chaff after (invalid IP options)	PASS
Small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IP fragments; interleave chaff before (invalid IP options)	PASS
Small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IP fragments; interleave chaff sandwich (invalid IP options); delay last fragment	PASS
Small TCP segments in random order; interleave chaff before (out-of-window sequence numbers); TCP MSS option; small IP fragments in random order; interleave chaff before (invalid IP options); delay random fragment	PASS
Small TCP segments in random order; interleave chaff sandwich (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment; small IP fragments	PASS
Small overlapping TCP segments; small fragments	PASS
Small overlapping TCP segments; delay last segment; small fragments; delay last fragment	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid length)	PASS
Small TCP segments; interleave chaff (invalid IP options; reserved flags set)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points before first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points past last address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points to middle of first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; more than two loose source route options)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points before first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points past last address))	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points to middle of first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; more than two strict source route options)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid timestamp option overflow_flag field)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid timestamp option pointer points past last address)	PASS
Small TCP segments; interleave chaff (invalid IP options; reserved flags set)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points before first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points past last address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points to middle of first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; more than two loose source route options)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points before first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points past last address))	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points to middle of first address)	PASS

Small TCP segments; interleave chaff (invalid IP options; more than two strict source route options)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid timestamp option overflow_flag field)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid timestamp option pointer points past last address)	PASS
Small TCP segments; interleave chaff (invalid IP options; reserved flags set)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points before first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points past last address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points to middle of first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; more than two loose source route options)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points before first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points past last address))	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points to middle of first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; more than two strict source route options)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid timestamp option overflow_flag field)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid timestamp option pointer points past last address)	PASS
Small TCP segments; interleave chaff (invalid IP options; reserved flags set)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points before first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points past last address)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points to middle of first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; more than two loose source route options)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points before first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points past last address))	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points to middle of first address)	PASS
Small TCP segments; interleave chaff (invalid IP options; more than two strict source route options)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid timestamp option overflow_flag field)	PASS
Small TCP segments; interleave chaff (invalid IP options; invalid timestamp option pointer points past last address)	PASS
IPv6 Packet Fragmentation/TCP Segmentation	
Small IPv6 fragments	PASS
Small IPv6 fragments in reverse order	PASS
Small IPv6 fragments in random order	PASS
Small IPv6 fragments; delay first fragment	PASS
Small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay first fragment	PASS
Small IPv6 fragments in reverse order; delay last fragment	PASS
Small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay random fragment	PASS
Small IPv6 fragments in random order; delay first fragment	PASS
Small IPv6 fragments in random order; delay last fragment	PASS
Small IPv6 fragments in random order; delay random fragment	PASS
Small IPv6 fragments; chunked	PASS
Small IPv6 fragments in reverse order; chunked	PASS

Small IPv6 fragments in random order; chunked	PASS
Small IPv6 fragments; delay first fragment; chunked	PASS
Small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay first fragment; chunked	PASS
Small TCP segments	PASS
Small TCP segments in reverse order	PASS
Small TCP segments in random order	PASS
Small TCP segments; delay first segment	PASS
Small TCP segments in reverse order; delay last segment	PASS
Small TCP segments; interleave chaff (invalid TCP checksums); delay first segment	PASS
Small TCP segments in random order; interleave chaff after (older PAWS timestamps); delay last segment	PASS
Small TCP segments in random order; interleave chaff before (out-of-window sequence numbers); TCP MSS option	PASS
Small TCP segments in random order; interleave chaff sandwich (requests to resynch sequence numbers mid-stream); TCP window scale option	PASS
Small TCP segments in random order; interleave chaff before (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment	PASS
Small overlapping TCP segments	PASS
Small TCP segments; chunked	PASS
Small TCP segments in reverse order; chunked	PASS
Small TCP segments in random order; chunked	PASS
Small TCP segments; delay first segment; chunked	PASS
Small TCP segments in reverse order; delay last segment; chunked	PASS
Small overlapping TCP segments; chunked	PASS
Small overlapping TCP segments; chunked; compressed with gzip	PASS
Small overlapping TCP segments; chunked; compressed with deflate	PASS
Small TCP segments; small IPv6 fragments	PASS
Small TCP segments; small IPv6 fragments in reverse order	PASS
Small TCP segments in random order; small IPv6 fragments	PASS
Small TCP segments; small IPv6 fragments in random order	PASS
Small TCP segments in random order; small IPv6 fragments in reverse order	PASS
Small TCP segments in random order; interleave chaff before (invalid TCP checksums); small IPv6 fragments in reverse order	PASS
Small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IPv6 fragments	PASS
Small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IPv6 fragments; delay last fragment	PASS
Small TCP segments in random order; interleave chaff sandwich (out-of-window sequence numbers); small IPv6 fragments in random order; delay random fragment	PASS
Small TCP segments in random order; interleave chaff after (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment; small IPv6 fragments	PASS
Small overlapping TCP segments; small IPv6 fragments	PASS
Small overlapping TCP segments; delay last segment; small IPv6 fragments; delay last fragment	PASS
Small TCP segments; small IPv6 fragments; chunked	PASS
Small TCP segments; small IPv6 fragments in reverse order; chunked	PASS
Small TCP segments in random order; small IPv6 fragments; chunked	PASS
Small TCP segments; small IPv6 fragments in random order; chunked	PASS
Small TCP segments in random order; small IPv6 fragments in reverse order; chunked	PASS

Small overlapping TCP segments; small IPv6 fragments; chunked	PASS
Small overlapping TCP segments; delay last segment; small IPv6 fragments; delay last fragment; chunked	PASS
Small TCP segments; small IPv6 fragments; chunked; compressed with gzip	PASS
Small TCP segments; small IPv6 fragments in reverse order; chunked; compressed with gzip	PASS
Small TCP segments in random order; small IPv6 fragments; chunked; compressed with gzip	PASS
Small TCP segments; small IPv6 fragments in random order; chunked; compressed with gzip	PASS
Small TCP segments in random order; small IPv6 fragments in reverse order; chunked; compressed with gzip	PASS
Small overlapping TCP segments; small IPv6 fragments; chunked; compressed with gzip	PASS
Small overlapping TCP segments; delay last segment; small IPv6 fragments; delay last fragment; chunked; compressed with gzip	PASS
Small TCP segments; small IPv6 fragments; chunked; compressed with deflate	PASS
Small TCP segments; small IPv6 fragments in reverse order; chunked; compressed with deflate	PASS
Small TCP segments in random order; small IPv6 fragments; chunked; compressed with deflate	PASS
Small TCP segments; small IPv6 fragments in random order; chunked; compressed with deflate	PASS
Small TCP segments in random order; small IPv6 fragments in reverse order; chunked; compressed with deflate	PASS
Small overlapping TCP segments; small IPv6 fragments; chunked; compressed with deflate	PASS
Small overlapping TCP segments; delay last segment; small IPv6 fragments; delay last fragment; chunked; compressed with deflate	PASS
HTML Evasions (*included in exploit block rate calculations)	
js-binary-obfuscation*	FAIL
babel-minify*	FAIL
Closure*	FAIL
code-protect*	PASS
Confusion*	PASS
Jfogs*	PASS
jfogs-reverse*	PASS
Jjencode*	FAIL
Jsbeautifier*	FAIL
Jsmint*	FAIL
js-obfuscator*	PASS
qzx-obfuscator*	PASS
js-binary-obfuscation; chunked*	FAIL
babel-minify; chunked*	FAIL
closure; chunked*	FAIL
code-protect; chunked*	PASS
confusion; chunked*	PASS
jfogs; chunked*	PASS
jfogs-reverse; chunked*	PASS
jjencode; chunked*	FAIL
jsbeautifier; chunked*	FAIL
jsmin; chunked*	FAIL
js-obfuscator; chunked*	PASS
qzx-obfuscator; chunked*	PASS

Chunked and gzip compressed js-binary-obfuscation*	FAIL
Chunked and gzip compressed babel-minify*	FAIL
Chunked and gzip compressed closure*	FAIL
Chunked and gzip compressed code-protect*	PASS
Chunked and gzip compressed confusion*	PASS
Chunked and gzip compressed jfogs*	PASS
Chunked and gzip compressed jfogs-reverse*	PASS
Chunked and gzip compressed jjencode*	FAIL
Chunked and gzip compressed jsbeautifier*	FAIL
Chunked and gzip compressed jsmin*	FAIL
Chunked and gzip compressed js-obfuscator*	PASS
Chunked and gzip compressed qzx-obfuscator*	PASS
Chunked and deflate compressed js-binary-obfuscation*	FAIL
Chunked and deflate compressed babel-minify*	FAIL
Chunked and deflate compressed closure*	FAIL
Chunked and deflate compressed code-protect*	PASS
Chunked and deflate compressed confusion*	PASS
Chunked and deflate compressed jfogs*	PASS
Chunked and deflate compressed jfogs-reverse*	PASS
Chunked and deflate compressed jjencode*	FAIL
Chunked and deflate compressed jsbeautifier*	FAIL
Chunked and deflate compressed jsmin*	FAIL
Chunked and deflate compressed js-obfuscator*	PASS
Chunked and deflate compressed qzx-obfuscator*	PASS
UTF-8 encoding	PASS
UTF-8 encoding with BOM	PASS
UTF-16 encoding with BOM	PASS
UTF-8 encoding; no http or html declarations	PASS
UTF-8 encoding with BOM; no http or html declarations	PASS
UTF-16 encoding with BOM; no http or html declarations	PASS
UTF-16-LE encoding without BOM	PASS
UTF-16-BE encoding without BOM	PASS
UTF-16-LE encoding without BOM; no http or html declarations	PASS
UTF-16-BE encoding without BOM; no http or html declarations	PASS
UTF-7 encoding	PASS
UTF-8 encoding; chunked	PASS
UTF-8 encoding with BOM; chunked	PASS
UTF-16 encoding with BOM; chunked	PASS
UTF-8 encoding; no http or html declarations; chunked	PASS
UTF-8 encoding with BOM; no http or html declarations; chunked	PASS
UTF-16 encoding with BOM; no http or html declarations; chunked	PASS
UTF-16-LE encoding without BOM; chunked	PASS

UTF-16-BE encoding without BOM; chunked	PASS
UTF-16-LE encoding without BOM; no http or html declarations; chunked	PASS
UTF-16-BE encoding without BOM; no http or html declarations; chunked	PASS
UTF-7 encoding; chunked	PASS
UTF-16-LE encoding without BOM; no http or html declarations; chunked and gzip compressed	PASS
UTF-16-BE encoding without BOM; no http or html declarations; chunked and gzip compressed	PASS
UTF-16-LE encoding without BOM; no http or html declarations; chunked and deflate compressed	PASS
UTF-16-BE encoding without BOM; no http or html declarations; chunked and deflate compressed	PASS
EICAR string included at top of HTML; comments removed	PASS
Hex-encoded script decoded using JavaScript unescape*	PASS
Unicode-encoded script decoded using JavaScript unescape*	PASS
Hex-encoded script as variable decoded using JavaScript unescape*	PASS
Unicode-encoded script as variable decoded using JavaScript unescape*	PASS
Hex-encoded script decoded using JavaScript unescape; chunked*	PASS
Unicode-encoded script decoded using JavaScript unescape; chunked*	PASS
Hex-encoded script as variable decoded using JavaScript unescape; chunked*	PASS
Unicode-encoded script as variable decoded using JavaScript unescape; chunked*	PASS
Padded with <=5MB	PASS
Padded with <=25MB	PASS
Padded with >25MB	PASS
Padded with <=5MB	PASS
Padded with <=25MB	PASS
Padded with >25MB	PASS
Padded with <=5MB; chunked	PASS
Padded with <=25MB; chunked	PASS
Padded with >25MB; chunked	PASS
Padded with <=5MB; chunked	PASS
Padded with <=25MB; chunked	PASS
Padded with >25MB; chunked	PASS
Padded with <=5MB; chunked and compressed with gzip	PASS
Padded with <=25MB; chunked and compressed with gzip	PASS
Padded with >25MB; chunked and compressed with gzip	PASS
Padded with <=5MB; chunked and compressed with gzip	PASS
Padded with <=25MB; chunked and compressed with gzip	PASS
Padded with >25MB; chunked and compressed with gzip	PASS
Padded with <=5MB; chunked and compressed with deflate	PASS
Padded with <=25MB; chunked and compressed with deflate	PASS
Padded with >25MB; chunked and compressed with deflate	PASS
Padded with <=5MB; chunked and compressed with deflate	PASS
Padded with <=25MB; chunked and compressed with deflate	PASS
Padded with >25MB; chunked and compressed with deflate	PASS

Resiliency (*included in exploit block rate calculations)	
Attack on nonstandard ports	PASS
Base exploit; chunked*	PASS
Base exploit; chunked and gzip compressed*	PASS
Base exploit; chunked and deflate compressed*	PASS
External VBScript file loaded from HTML*	PASS
Multiple VBScript files loaded from HTML*	PASS
Multiple VBScript files loaded with external JavaScript file*	PASS
External VBScript file loaded from HTML; chunked*	PASS
Multiple VBScript files loaded from HTML; chunked*	PASS
Multiple VBScript files loaded with external JavaScript file; chunked*	PASS
External VBScript file loaded from HTML; chunked and gzip compressed*	PASS
Multiple VBScript files loaded from HTML; chunked and gzip compressed*	PASS
Multiple VBScript files loaded with external JavaScript file; chunked and gzip compressed*	PASS
External VBScript file loaded from HTML; chunked and deflate compressed*	PASS
Multiple VBScript files loaded from HTML; chunked and deflate compressed*	PASS
Multiple VBScript files loaded with external JavaScript file; chunked and deflate compressed*	PASS
VBScript interspersed randomly with null bytes; content="IE=10" replaced with content="IE=EmulateIE8"*	FAIL
VBScript interspersed randomly with null bytes; content="IE=10" replaced with content="IE=EmulateIE8"; chunked*	FAIL
VBScript interspersed randomly with null bytes; content="IE=10" replaced with content="IE=EmulateIE8"; chunked and gzip compressed*	FAIL
VBScript interspersed randomly with null bytes; content="IE=10" replaced with content="IE=EmulateIE8"; chunked and deflate compressed*	FAIL
Veil Ordnance generated bind shell stager*	PASS
msfvenom generated bind shell stager*	PASS
msfvenom generated bind shell stager; shikata_ga_nai encoded 10x*	PASS
msfvenom generated bind shell stager; call4_dword_xor encoded 10x*	PASS
Both spaces and linefeeds replaced with multiples of each*	PASS
Reorder function definitions*	PASS
Rename variables and functions*	PASS
Numeric values/equations modified and/or inserted; hexadecimal values replaced with decimal values*	PASS
Numeric values/equations modified and/or resolved*	PASS
Numeric values/equations modified and/or resolved; numeric values/equations modified and/or inserted; hexadecimal values replaced with decimal values*	PASS
Some strings split with "+" and "&"; some lines split with "_ "*	PASS
Some strings split with "+" and "&"*	PASS
Some lines split with "_ "*	PASS
Change all chr() to chrw() and vice versa where possible*	PASS
Change chr() and chrw() to chrb()*	PASS
Some script commands/strings converted to series of chr()/CIng*	FAIL
Some script commands/strings converted to series of chr()*	FAIL
Change chr() and chrw() to chrb(); Some script commands/strings converted to series of chr(); change all chr() to chrw() and vice versa where possible*	FAIL

Change chr() and chrw() to chrb(); Some script commands/strings converted to series of chr()/Cln; change all chr() to chrw() and vice versa where possible*	FAIL
Combination of res-ord-001; res-wsp-001*	PASS
Combination of res-ren-001; res-wsp-001*	PASS
Combination of res-ren-001; res-ord-001*	PASS
Combination of res-ren-001; res-ord-001; res-wsp-001*	PASS
Combination of res-ren-001; res-ord-001; res-wsp-001; res-mth-003*	PASS
Combination of res-ren-001; res-ord-001; res-wsp-001; res-mth-003; res-spl-001*	FAIL
Combination of res-ren-001; res-ord-001; res-mth-003; res-wsp-001; res-pay-001*	PASS
Combination of res-ren-001; res-ord-001; res-wsp-001; res-mth-003; res-spl-001; res-chr-006*	FAIL
Combination of res-ren-001; res-ord-001; res-wsp-001; res-mth-003; res-spl-001; res-chr-005*	FAIL
Combination of res-ren-001; res-ord-001; res-wsp-001; res-mth-003; res-spl-001; res-pay-001*	FAIL
Combination of res-ren-001; res-ord-001; res-wsp-001; res-mth-003; res-spl-001; res-chr-006; res-pay-001*	FAIL
Combination of res-ren-001; res-ord-001; res-wsp-001; res-mth-003; res-spl-001; res-chr-005; res-pay-001*	FAIL
Combination of res-nb-001; res-ren-001; res-ord-001; res-wsp-001; res-spl-001; res-chr-006; res-pay-001*	FAIL
Combination of res-nb-001; res-ren-001; res-ord-001; res-wsp-001; res-spl-001; res-chr-005; res-pay-001*	FAIL
Combination of res-nb-001; res-ren-001; res-ord-001; res-wsp-001; res-spl-001; res-chr-006; res-pay-001; chunked*	FAIL
Combination of res-nb-001; res-ren-001; res-ord-001; res-wsp-001; res-spl-001; res-chr-005; res-pay-001; chunked*	FAIL
Combination of res-nb-001; res-ren-001; res-ord-001; res-wsp-001; res-spl-001; res-chr-006; res-pay-001; chunked and gzip compressed*	FAIL
Combination of res-nb-001; res-ren-001; res-ord-001; res-wsp-001; res-spl-001; res-chr-005; res-pay-001; chunked and gzip compressed*	FAIL
Combination of res-nb-001; res-ren-001; res-ord-001; res-wsp-001; res-spl-001; res-chr-006; res-pay-001; chunked and deflate compressed*	FAIL
Combination of res-nb-001; res-ren-001; res-ord-001; res-wsp-001; res-spl-001; res-chr-005; res-pay-001; chunked and deflate compressed*	FAIL
Combination of Evasions	
UTF-8 encoding; HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'); small TCP segments; small IP fragments; padding	PASS
UTF-8 encoding with BOM; HTTP/1.1 chunked response with chunk sizes followed by end of transmission (hex '04'); small TCP segments; small IP fragments in reverse order; padding	PASS
UTF-16 encoding with BOM; HTTP/1.1 chunked response with chunk sizes followed by end of transmission block (hex '17'); small TCP segments in random order; small IP fragments; padding	PASS
UTF-8 encoding; no http or html declarations; HTTP/1.1 chunked response with chunk sizes followed by file separator (hex '1c'); small TCP segments; small IP fragments in random order; padding	PASS
UTF-8 encoding with BOM; no http or html declarations; HTTP/1.1 chunked response with chunk sizes followed by comma (hex '2c'); small TCP segments in random order; small IP fragments in reverse order; padding	PASS
UTF-16 encoding with BOM; no http or html declarations; HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a \$ (hex '24'); small TCP segments; small IPv6 fragments; padding	PASS
UTF-16-LE encoding without BOM; HTTP/1.1 chunked response with final chunk size of '00' (rather than '0'); small TCP segments; small IPv6 fragments in reverse order; padding	PASS
UTF-16-BE encoding without BOM; 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; small TCP segments in random order; small IPv6 fragments; padding	PASS
UTF-16-LE encoding without BOM; no http or html declarations; 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; small TCP segments; small IPv6 fragments in random order; padding	PASS
UTF-16-BE encoding without BOM; no http or html declarations; HTTP/1.0 response declaring chunking; served without chunking; small TCP segments in random order; small IPv6 fragments in reverse order; padding	PASS

UTF-7 encoding; HTTP/1.0 response declaring chunking with invalid content-length header; served without chunking; small TCP segments in random order; small IPv6 fragments in reverse order; padding	PASS
HTTP/1.1 response declaring chunking; 16-byte segments break CRLF at end of response status header; served unchunked	PASS
HTTP/1.1 response declaring chunking and gzip compression; 16-byte segments break CRLF at end of response status header; served unchunked and compressed	PASS
HTTP/1.1 response declaring gzip compression as "HTTP/1.1 200 OK\rContent-Encoding: gzip\r\n"; 16 byte segments break CRLF at end of response status header; served uncompressed	PASS
HTTP/1.1 response declaring chunking and gzip compression as "HTTP/1.1 200 OK\rContent-Encoding: gzip\r\n"; 16-byte segments break CRLF at end of response status header; served unchunked and uncompressed	PASS
HTTP/1.1 response declaring gzip compression and chunking as "HTTP/1.1 200 OK\rTransfer-Encoding: chunked\r\n"; 16-byte segments break CRLF at end of response status header; served unchunked and compressed	PASS
HTTP/1.0 200 OK\r/1.1\n status response declaring chunking; 16-byte segments break CRLF at end of response status header; served chunked	PASS
UTF-16-BE encoding without BOM; 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'); chunk with some data in chunk-extension field; small TCP segments in random order; small IPv6 fragments; padding	PASS
UTF-7 encoding; HTTP/1.0 response declaring chunking with invalid content-length header; chunk with some data in chunk-extension field; small TCP segments in random order; small IPv6 fragments in reverse order; padding	PASS
HTTP/1.1 response declaring chunking; 16-byte segments break CRLF at end of response status header; chunk with some data in chunk-extension field	PASS
HTTP/1.1 response declaring chunking and gzip compression as "HTTP/1.1 200 OK\rContent-Encoding: gzip\r\n"; 16-byte segments break CRLF at end of response status header; served uncompressed; chunk with some data in chunk-extension field	PASS
RPC Fragmentation	
One-byte fragmentation (ONC)	PASS
Two-byte fragmentation (ONC)	PASS
All fragments including Last Fragment (LF) will be sent in one TCP segment (ONC)	PASS
All fragments except Last Fragment (LF) will be sent in one TCP segment. LF will be sent in separate TCP segment (ONC).	PASS
One RPC fragment will be sent per TCP segment (ONC)	PASS
One LF split over more than one TCP segment. In this case no RPC fragmentation is performed (ONC).	PASS
Canvas Reference Implementation Level 1 (MS)	PASS
Canvas Reference Implementation Level 2 (MS)	PASS
Canvas Reference Implementation Level 3 (MS)	PASS
Canvas Reference Implementation Level 4 (MS)	PASS
Canvas Reference Implementation Level 5 (MS)	PASS
Canvas Reference Implementation Level 6 (MS)	PASS
Canvas Reference Implementation Level 7 (MS)	PASS
Canvas Reference Implementation Level 8 (MS)	PASS
Canvas Reference Implementation Level 9 (MS)	PASS
Canvas Reference Implementation Level 10 (MS)	PASS
URL Obfuscation	
URL encoding – Level 1 (minimal)	PASS
URL encoding – Level 2	PASS
URL encoding – Level 3	PASS
URL encoding – Level 4	PASS
URL encoding – Level 5	PASS
URL encoding – Level 6	PASS

URL encoding – Level 7		PASS
URL encoding – Level 8 (extreme)		PASS
Directory Insertion		PASS
Premature URL ending		PASS
Long URL		PASS
Fake parameter		PASS
TAB separation		PASS
Case sensitivity		PASS
Windows \ delimiter		PASS
Session splicing		PASS
FTP/Telnet Evasion		
Inserting spaces in FTP command lines		PASS
Inserting non-text Telnet opcodes – Level 1 (minimal)		PASS
Inserting non-text Telnet opcodes – Level 2		PASS
Inserting non-text Telnet opcodes – Level 3		PASS
Inserting non-text Telnet opcodes – Level 4		PASS
Inserting non-text Telnet opcodes – Level 5		PASS
Inserting non-text Telnet opcodes – Level 6		PASS
Inserting non-text Telnet opcodes – Level 7		PASS
Inserting non-text Telnet opcodes – Level 8 (extreme)		PASS
Performance		
Raw Packet Processing Performance (UDP Traffic)		Weighting for HTTP NSS-Tested Throughput
64-Byte Packets		Mbps
64-Byte Packets	0%	17,610
128-Byte Packets	1%	18,160
256-Byte Packets	1%	18,610
512-Byte Packets	1%	18,810
1,024-Byte Packets	3%	20,000
1,280-Byte Packets	1%	20,000
1,514-Byte Packets	3%	20,000
Latency – UDP		Microseconds
64-Byte Packets		12
128-Byte Packets		13
256-Byte Packets		13
512-Byte Packets		14
1,024-Byte Packets		17
1,280-Byte Packets		23
1,514-Byte Packets		24
Maximum Capacity		CPS
Theoretical Max. Concurrent TCP Connections		4,193,280
Maximum TCP Connections per Second		62,500
Maximum HTTP Connections per Second		44,180
Maximum HTTP Transactions per Second		73,060

HTTP Capacity		Weighting for HTTP NSS-Tested Throughput		CPS
2,500 Connections per Second – 44 KB Response		8%		25,000
5,000 Connections per Second – 21 KB Response		8%		40,420
10,000 Connections per Second – 10 KB Response		7%		42,470
20,000 Connections per Second – 4.5 KB Response		7%		42,600
40,000 Connections per Second – 1.7 KB Response		4%		43,390
Application Average Response Time – HTTP (at 90% Max Load)				Milliseconds
2,500 Connections per Second – 44 KB Response				1.55
5,000 Connections per Second – 21 KB Response				1.18
10,000 Connections per Second – 10 KB Response				1.15
20,000 Connections per Second – 4.5 KB Response				1.11
40,000 Connections per Second – 1.7 KB Response				6.40
HTTP Capacity with HTTP Persistent Connections				CPS
250 Connections per Second				985
500 Connections per Second				1,718
1,000 Connections per Second				2,740
HTTP Capacity with Bidirectional HTTP Persistent Connections				CPS
250 Connections per Second				1,021
500 Connections per Second				1,738
1,000 Connections per Second				2,878
Single Application Flows		Weighting for HTTP NSS-Tested Throughput		Mbps
Telephony		17%		9,904
Financial		0%		4,861
Email		12%		2,616
File Sharing		7%		10,000
Fileserver		0%		1,103
Remote Console		1%		1,228
Video		16%		6,381
Database		3%		4,477
SSL/TLS Functionality Testing		Decryption	Layer 3 Policy	Layer 4 Policy
TLS ECDHE RSA WITH AES 256 GCM SHA384		PASS	PASS	PASS
TLS ECDHE RSA WITH AES 128 GCM SHA256		PASS	PASS	PASS
TLS ECDHE ECDSA WITH AES 128 GCM SHA256		PASS	PASS	PASS
TLS ECDHE RSA WITH AES 256 CBC SHA384		PASS	PASS	PASS
TLS DHE RSA WITH AES 256 GCM SHA384		PASS	PASS	PASS
TLS ECDHE RSA WITH AES 256 CBC SHA		PASS	PASS	PASS
TLS DHE RSA WITH AES 256 CBC SHA		PASS	PASS	PASS
TLS RSA WITH AES 256 CBC SHA		PASS	PASS	PASS
TLS RSA WITH AES 128 CBC SHA		PASS	PASS	PASS
TLS RSA WITH AES 256 CBC SHA256		PASS	PASS	PASS
TLS RSA WITH AES 256 GCM SHA384		PASS	PASS	PASS
TLS ECDHE RSA WITH AES 128 CBC SHA256		PASS	PASS	PASS

TLS RSA WITH AES 128 CBC SHA256	PASS	PASS	PASS
TLS RSA WITH AES 128 GCM SHA256	PASS	PASS	PASS
TLS ECDHE RSA WITH AES 128 CBC SHA	PASS	PASS	PASS
TLS ECDHE ECDSA WITH AES 256 GCM SHA384	PASS	PASS	PASS
TLS DHE RSA WITH AES 128 CBC SHA	PASS	PASS	PASS
TLS DHE RSA WITH AES 128 GCM SHA256	PASS	PASS	PASS
TLS DHE RSA WITH AES 256 CBC SHA256	PASS	PASS	PASS
TLS DHE RSA WITH CAMELLIA 256 CBC SHA	FAIL	FAIL	FAIL
TLS DHE RSA WITH SEED CBC SHA	FAIL	FAIL	FAIL
TLS RSA WITH SEED CBC SHA	FAIL	FAIL	FAIL
TLS RSA WITH CAMELLIA 256 CBC SHA	FAIL	FAIL	FAIL
TLS DHE RSA WITH AES 128 CBC SHA256	PASS	PASS	PASS
SSL/TLS Functionality Testing	SNI Policy	CN/SAN Policy	Revoked Certificate
TLS ECDHE RSA WITH AES 256 GCM SHA384	PASS	PASS	PASS
TLS ECDHE RSA WITH AES 128 GCM SHA256	PASS	PASS	PASS
TLS ECDHE ECDSA WITH AES 128 GCM SHA256	PASS	PASS	PASS
TLS ECDHE RSA WITH AES 256 CBC SHA384	PASS	PASS	PASS
TLS DHE RSA WITH AES 256 GCM SHA384	PASS	PASS	PASS
TLS ECDHE RSA WITH AES 256 CBC SHA	PASS	PASS	PASS
TLS DHE RSA WITH AES 256 CBC SHA	PASS	PASS	PASS
TLS RSA WITH AES 256 CBC SHA	PASS	PASS	PASS
TLS RSA WITH AES 128 CBC SHA	PASS	PASS	PASS
TLS RSA WITH AES 256 CBC SHA256	PASS	PASS	PASS
TLS RSA WITH AES 256 GCM SHA384	PASS	PASS	PASS
TLS ECDHE RSA WITH AES 128 CBC SHA256	PASS	PASS	PASS
TLS RSA WITH AES 128 CBC SHA256	PASS	PASS	PASS
TLS RSA WITH AES 128 GCM SHA256	PASS	PASS	PASS
TLS ECDHE RSA WITH AES 128 CBC SHA	PASS	PASS	PASS
TLS ECDHE ECDSA WITH AES 256 GCM SHA384	PASS	PASS	PASS
TLS DHE RSA WITH AES 128 CBC SHA	PASS	PASS	PASS
TLS DHE RSA WITH AES 128 GCM SHA256	PASS	PASS	PASS
TLS DHE RSA WITH AES 256 CBC SHA256	PASS	PASS	PASS
TLS DHE RSA WITH CAMELLIA 256 CBC SHA	FAIL	FAIL	FAIL
TLS DHE RSA WITH SEED CBC SHA	FAIL	FAIL	FAIL
TLS RSA WITH SEED CBC SHA	FAIL	FAIL	FAIL
TLS RSA WITH CAMELLIA 256 CBC SHA	FAIL	FAIL	FAIL
TLS DHE RSA WITH AES 128 CBC SHA256	PASS	PASS	PASS
SSL/TLS Functionality Testing	Expired Certificate	Session ID	Session Ticket
TLS ECDHE RSA WITH AES 256 GCM SHA384	PASS	PASS	PASS
TLS ECDHE RSA WITH AES 128 GCM SHA256	PASS	PASS	PASS
TLS ECDHE ECDSA WITH AES 128 GCM SHA256	PASS	PASS	PASS
TLS ECDHE RSA WITH AES 256 CBC SHA384	PASS	PASS	PASS

TLS DHE RSA WITH AES 256 GCM SHA384	PASS	PASS	PASS	
TLS ECDHE RSA WITH AES 256 CBC SHA	PASS	PASS	PASS	
TLS DHE RSA WITH AES 256 CBC SHA	PASS	PASS	PASS	
TLS RSA WITH AES 256 CBC SHA	PASS	PASS	PASS	
TLS RSA WITH AES 128 CBC SHA	PASS	PASS	PASS	
TLS RSA WITH AES 256 CBC SHA256	PASS	PASS	PASS	
TLS RSA WITH AES 256 GCM SHA384	PASS	PASS	PASS	
TLS ECDHE RSA WITH AES 128 CBC SHA256	PASS	PASS	PASS	
TLS RSA WITH AES 128 CBC SHA256	PASS	PASS	PASS	
TLS RSA WITH AES 128 GCM SHA256	PASS	PASS	PASS	
TLS ECDHE RSA WITH AES 128 CBC SHA	PASS	PASS	PASS	
TLS ECDHE ECDSA WITH AES 256 GCM SHA384	PASS	PASS	PASS	
TLS DHE RSA WITH AES 128 CBC SHA	PASS	PASS	PASS	
TLS DHE RSA WITH AES 128 GCM SHA256	PASS	PASS	PASS	
TLS DHE RSA WITH AES 256 CBC SHA256	PASS	PASS	PASS	
TLS DHE RSA WITH CAMELLIA 256 CBC SHA	FAIL	FAIL	FAIL	
TLS DHE RSA WITH SEED CBC SHA	FAIL	FAIL	FAIL	
TLS RSA WITH SEED CBC SHA	FAIL	FAIL	FAIL	
TLS RSA WITH CAMELLIA 256 CBC SHA	FAIL	FAIL	FAIL	
TLS DHE RSA WITH AES 128 CBC SHA256	PASS	PASS	PASS	
SSL/TLS Performance				
Maximum HTTP(S) Connections per Second	Key Size	CPS		
TLS ECDHE RSA WITH AES 256 GCM SHA384	2,048-bit key	1,541		
TLS ECDHE RSA WITH AES 256 GCM SHA384	4,096-bit key	1,475		
TLS ECDHE RSA WITH AES 128 GCM SHA256	2,048-bit key	1,551		
TLS ECDHE ECDSA WITH AES 128 GCM SHA256	256-bit key	1,827		
TLS ECDHE RSA WITH AES 256 CBC SHA384	2,048-bit key	1,544		
No Encryption (Baseline)	Session	Response Size	CPS	Mbps
HTTP Capacity, No Persistence	Single HTTP GET Request	2,880 KB	400	10,000
		768 KB	1500	10,000
		192 KB	6000	10,000
		44 KB	21,353	8,541
HTTP Capacity with Persistent Connections	10 HTTP GET Requests	288 KB	400	10,000
		76.8 KB	1,500	10,000
		19.2 KB	3,419	5,698
		4.4 KB	5,741	2,296
TLS ECDHE RSA WITH AES 256 GCM SHA384 (2k)	Session	Response Size	Weighting for HTTPS (SSL/TLS) NSS-Tested Throughput	Mbps
HTTP Capacity, No Persistence	Single HTTPS GET Request	2,880 KB	191	4,775
		768 KB	511	3,407
		192 KB	1006	1,677
		44 KB	1,398	559

HTTP Capacity with Persistent Connections	10 HTTPS GET Requests	288 KB	9	225
		76.8 KB	473	3,153
		19.2 KB	812	1,353
		4.4 KB	1,115	446
TLS ECDHE RSA WITH AES 256 GCM SHA384 (4k)	Session	Response Size	Weighting for HTTPS (SSL/TLS) NSS-Tested Throughput	Mbps
HTTP Capacity, No Persistence	Single HTTPS GET Request	2880 KB	167	4,175
		768 KB	453	3,020
		192 KB	947	1,578
		44 KB	1,361	544
HTTP Capacity with Persistent Connections	10 HTTPS GET Requests	288 KB	13	325
		76.8 KB	354	2,360
		19.2 KB	822	1,370
		4.4 KB	1,013	405
TLS ECDHE RSA WITH AES 128 GCM SHA256 (2k)	Session	Response Size	Weighting for HTTPS (SSL/TLS) NSS-Tested Throughput	Mbps
HTTP Capacity, No Persistence	Single HTTPS GET Request	2,880 KB	191	4,775
		768 KB	509	3,393
		192 KB	1,014	1,690
		44 KB	1,249	500
HTTP Capacity with Persistent Connections	10 HTTPS GET Requests	288 KB	29	725
		76.8 KB	373	2,487
		19.2 KB	815	1,358
		4.4 KB	1,146	458

TLS ECDHE ECDSA WITH AES 128 GCM SHA256	Session	Response Size	Weighting for HTTPS (SSL/TLS) NSS-Tested Throughput	Mbps
HTTP Capacity, No Persistence	Single HTTPS GET Request	2,880 KB	200	5,000
		768 KB	559	3,727
		192 KB	1,222	2,037
		44 KB	1,714	686
HTTP Capacity with Persistent Connections	10 HTTPS GET Requests	288 KB	15	375
		76.8 KB	509	3,393
		19.2 KB	981	1,635
		4.4 KB	1,335	534
TLS ECDHE RSA WITH AES 256 CBC SHA384 (2k)	Session	Response Size	Weighting for HTTPS (SSL/TLS) NSS-Tested Throughput	Mbps
HTTP Capacity, No Persistence	Single HTTPS GET Request	2,880 KB	180	4,500
		768 KB	509	3,393
		192 KB	1,015	1,692
		44 KB	1,362	545
HTTP Capacity with Persistent Connections	10 HTTPS GET Requests	288 KB	24	600
		76.8 KB	307	2,047
		19.2 KB	817	1,362
		4.4 KB	1,145	458
Stability and Reliability				
Blocking under Extended Attack				PASS
Passing Legitimate Traffic under Extended Attack				PASS
Behavior of the State Engine under Load				
Attack Detection/Blocking – Normal Load				PASS
State Preservation – Normal Load				PASS
Pass Legitimate Traffic – Normal Load				PASS
State Preservation – Maximum Exceeded				PASS
Drop Traffic – Maximum Exceeded				PASS
Power Fail				PASS
Backup/Restore				PASS
Persistence of Data				PASS
Stability				PASS
Total Cost of Ownership				
Ease of Use				
Initial Setup (Hours)				8
Expected Costs				US\$
Initial Purchase (Hardware as Tested)				\$37,500
Installation Labor Cost (@\$75/hr)				\$600
Annual Cost of Maintenance and Support (Hardware/Software)				\$4,800
Annual Cost of Updates (IPS/AV/etc.)				\$6,000

Total Cost of Ownership	
Year 1	\$48,900
Year 2	\$10,800
Year 3	\$10,800
3-Year Total Cost of Ownership	\$70,500

Figure 31 – Detailed Scorecard

Test Methodology

NSS Labs Next Generation Firewall (NGFW) Test Methodology v9.0

NSS Labs SSL/TLS Performance Test Methodology v1.3

NSS Labs Evasions Test Methodology v1.1

Copies of the test methodologies are available at www.nsslabs.com.

Contact Information

3711 South Mopac Expressway
Building 1, Suite 400
Austin, TX 78746
info@nsslabs.com
www.nsslabs.com

This and other related documents are available at www.nsslabs.com. To receive a licensed copy or report misuse, please contact NSS Labs.

© 2019 NSS Labs, Inc. All rights reserved. No part of this publication may be reproduced, copied/scanned, stored on a retrieval system, e-mailed or otherwise disseminated or transmitted without the express written consent of NSS Labs, Inc. (“us” or “we”).

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.