NEXT GENERATION INTRUSION PREVENTION SYSTEM (NGIPS) TEST REPORT

Fortinet FortiGate-100F v6.0.2 build6215 (GA) (IPS engine 4.00045 and signature pack 14.00697)

OCTOBER 18, 2019

Authors – Thomas Williams, Matt Wheeler

Fortinet FortiGate-100F v6.0.2 build6215 (GA) (IPS engine 4.00045 and signature pack 14.00697)

Summary

This document provides test results for the Fortinet FortiGate-100F v6.0.2 build6215 GA (IPS engine 4.00045 and signature pack 14.00697). During the NSS Labs 2019 Next Generation Intrusion Prevention (NGIPS) Group Test, the Fortinet FortiGate-100F v6.0.2 build6215 GA failed to detect 30 evasions. This affected its placement in NSS' 2019 NGIPS Security Value Map (SVM)™.

After working closely with NSS, Fortinet updated its software and released IPS engine 4.00045 and signature pack 14.00697. The updated device was subjected to testing under the same NGIPS Test Methodology (v5.0) and appropriately handled all of the 494 evasions it was tested against. Furthermore, the FortiGate-100F improved its exploit block rate by 0.73% while seeing a marginal drop in performance (481 Mbps for IPv4 and 261 Mbps for IPv6).

Security	**Security Effectiveness**		
	Exploit Block Rate[1]	99.91%	
	• Exploit Library	99.89%	
	• Live Exploits	100.00%	
	• Script Obfuscation	100.00%	
	Evasions Blocked	494/494	
	Stability & Reliability	PASS	
	False Positives	PASS	
Performance/Functionality	**Performance**		
	NSS-Tested Throughput (IPv4)	3,084 Mbps	
	NSS-Tested Throughput (IPv6)	3,047 Mbps	
	Maximum Capacity	**CPS (IPv4)**	**CPS (IPv6)**
	Theoretical Max. Concurrent TCP Connections	280,738	265,826
	Max TCP Connections/Second	15,980	15,970
	Max HTTP Connections/Second	11,350	11,270
	Max HTTP Transactions/Second	27,580	25,230
	HTTP Capacity	**CPS (IPv4)**	**CPS (IPv6)**
	2,500 Connections per Second – 44 KB Response	3,343	3,449
	5,000 Connections per Second – 21 KB Response	4,996	4,781
	10,000 Connections per Second – 10 KB Response	7,042	6,611
	20,000 Connections per Second – 4.5 KB Response	8,757	8,234
	40,000 Connections per Second – 1.7 KB Response	10,470	9,960
Cost	**Total Cost of Ownership (TCO)**		
	3-Year TCO (US$)	$4,935	

The product was subjected to thorough testing based on the Next Generation Intrusion Prevention System (NGIPS) Test Methodology v5.0 and the Evasions Test Methodology v1.1 (available at 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.

Table of Contents

Security Effectiveness 5

False Positive Testing 5

Exploit Library 5

Coverage by Attack Vector 6

Coverage by Date 6

Coverage by Target Vendor 7

Script Obfuscation 7

Live Exploits 8

Resistance to Evasion Techniques 8

IP Packet Fragmentation 9

TCP Segmentation 9

HTTP Obfuscation & Compression 10

HTML Obfuscation 10

Protection Resiliency 10

Performance 12

Maximum Capacity 12

HTTP Capacity 12

Application Average Response Time – HTTP 13

Single Application Flows 14

Raw Packet Processing Performance (UDP Throughput) 15

Raw Packet Processing Performance (UDP Latency) 16

NSS-Tested Throughput 16

Stability and Reliability 17

Appendix A: Product Scorecard 18

Test Methodology 39

Contact Information 39

Table of Figures

Figure 1 – Number of Exploits Blocked 5
Figure 2 – Coverage by Attack Vector 6
Figure 3 – Product Coverage by Date 6
Figure 4 – Product Coverage by Target Vendor 7
Figure 5 – Script Obfuscation Attacks Blocked 7
Figure 6 – Live Attacks Blocked 8
Figure 7 –Overview of Evasion Scores 9
Figure 8 – Concurrency and Connection Rates 12
Figure 9 – HTTP Connections per Second and Capacity 13
Figure 10 – Average Application Response Time (Milliseconds) 13
Figure 11 – Single Application Flows 14
Figure 12 – Raw Packet Processing Performance (UDP Traffic) 15
Figure 13 – UDP Latency in Microseconds 16
Figure 14 – NSS-Tested Throughput (Mbps) 16
Figure 15 – Stability and Reliability Results 17
Figure 16 – Detailed Scorecard 38

Security Effectiveness

The threat landscape is evolving constantly; attackers are refining their strategies and increasing both the volume and complexity of their attacks. Enterprises now are having to defend against everyday cybercriminal attacks as well as targeted attacks and even the rare advanced persistent threats (APTs). As attacks have increased in both volume and sophistication, it has become increasingly complicated for an enterprise to monitor its network for abnormalities and emerging attack patterns and take preventative or responsive action.

For this reason, we test several types of attacks ranging from widespread day-to-day attacks and current threat actor campaigns to targeted attacks and advanced (modified, custom, evasions) attacks. In this test, we validated whether or not the NGIPS could protect against a wide range of threats and whether or not these products are providing enterprises with the protection they believe they are purchasing.

Our security effectiveness tests verify that the NGIPS is capable of blocking and logging threats accurately while remaining resistant to false positives. The signatures/filters/rules that trigger false positives were turned off in order to replicate an enterprise's experience when deploying the device. Testing leverages the deep expertise of NSS engineers who utilize multiple commercial, open-source, and proprietary tools to employ attack methods that are currently being used by cybercriminals and other threat actors. All tests in this section are completed with no background network load.

False Positive Testing

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

Exploit Library

NSS' security effectiveness testing leverages the deep expertise of our engineers who utilize multiple commercial, open-source, and proprietary tools, including NSS' network live stack test environment as appropriate. With 1,783 exploits, this is the industry's most comprehensive test to date. Most notably, all of the exploits and payloads in this test have been validated such that:

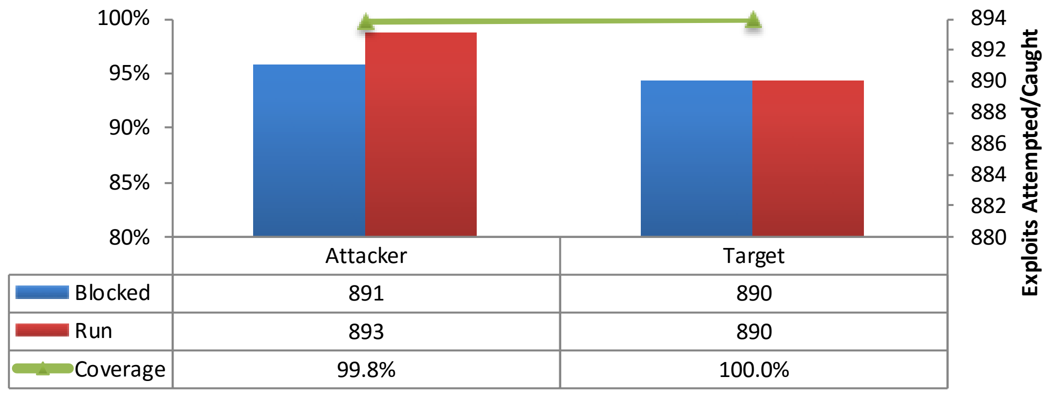
- A reverse shell is returned
- A bind shell is opened on the target, allowing the attacker to execute arbitrary commands
- Arbitrary code is executed
- A malicious payload is installed
- A system is rendered unresponsive
- Etc.

Product	Total Number of Exploits Run	Total Number of Exploits Blocked	Block Percentage
Fortinet FortiGate-100F v6.0.2 build6215 (GA) (IPS engine 4.00045 and signature pack 14.00697)	1,783	1,781	99.89%

Figure 1 – Number of Exploits Blocked

Coverage by Attack Vector

Because a failure to block attacks could result in significant compromise and severely impact critical business systems, network intrusion prevention systems should be evaluated against a broad set of exploits. Exploits can be categorized as either *attacker-initiated* or *target-initiated*. Attacker-initiated exploits are 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.



	Attacker	Target
🟦 Blocked	891	890
🟥 Run	893	890
▲ Coverage	99.8%	100.0%

Figure 2 – Coverage by Attack Vector

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.

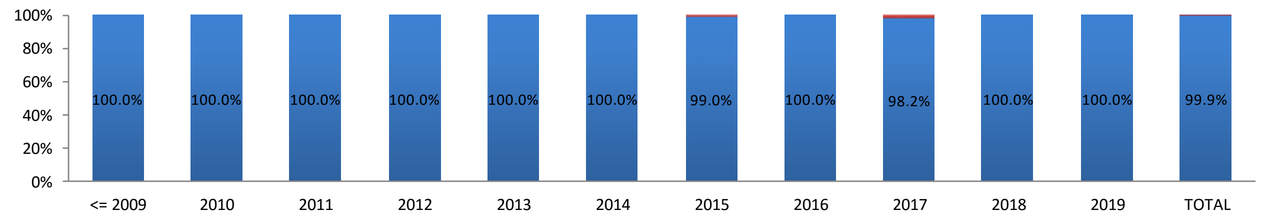


Figure 3 – Product Coverage by Date

Coverage by Target Vendor

Exploits within the *NSS Exploit Library* target a wide range of protocols and applications. Figure 4 depicts the coverage offered by the 100F for some of the top vendor targets represented for this round of testing.

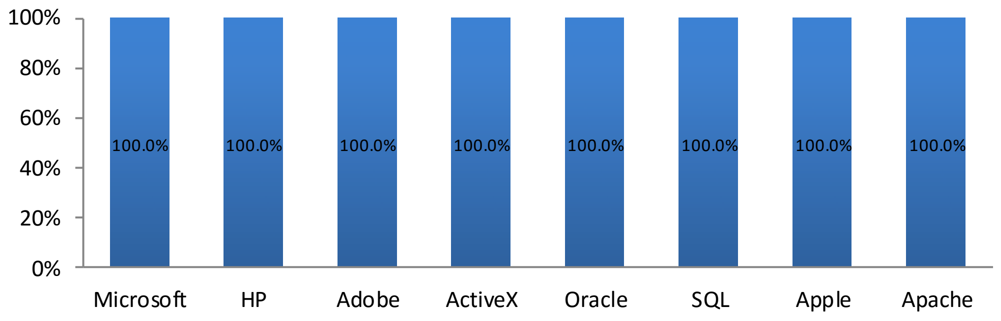


Figure 4 – Product Coverage by Target Vendor

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. 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
Fortinet FortiGate-100F v6.0.2 build6215 (GA) (IPS engine 4.00045 and signature pack 14.00697)	132	132	100.00%

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.[2]

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.

Product	Block Percentage
Fortinet FortiGate-100F v6.0.2 build6215 (GA) (IPS engine 4.00045 and signature pack 14.00697)	100.00%

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 solutions. 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.

The more classes of evasion that are missed (such as HTTP evasions, IP packet fragmentation, TCP stream segmentation and HTML obfuscation), the less effective the device. For example, it is better to miss all techniques in one evasion category, such as HTTP 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 TCP stream segmentation) have a greater impact on security effectiveness than those operating at the upper layers (HTTP evasions or HTML obfuscation). Many of the techniques used in this test have been widely known for years and should be considered minimum requirements.

Each evasion used active exploits (i.e., no pcaps). If an evasion evaded a victim machine's protections, it popped a shell on the victim machine. Victim machines in the test harness did not have endpoints installed.

While all devices were tested against 626 evasions, only 494 of these were used to calculate products' Evasions Blocked totals. Script obfuscations and resiliency evasions were not included in this total but were included in block rate calculations. This is because these types of attacks are considered "complex evasions" (HTML/JavaScript/VBScript) and require real-time code analysis in order to determine whether a function is legitimate or obfuscating an attack. Please see the individual Test Reports for details (available at nsslabs.com).

The FortiGate-100F blocked 464/494 evasions. Figure 7 provides an overview of the device's evasion scores.

[2] See the NSS Continuous Security Validation Platform for more details.

Test Procedure	Result
HTTP Evasions	PASS
IP Packet Fragmentation/TCP Segmentation	PASS
HTML Evasions[3]	PASS
Resiliency[4]	*See footnote*
Attacks on Nonstandard Ports	PASS
Combination of Evasions	PASS

Figure 7 –Overview of Evasion Scores

IP Packet Fragmentation

The Internet uses the Internet Protocol (IP) to transmit and route traffic from one computer to another. IP is connectionless, meaning that it transmits data to a remote host without knowing whether or not the host is ready to exchange the data. IP does not have any error detection/correction facility, and it does not guarantee the receipt of the datagrams.

There is always a possibility that a datagram will be lost or corrupted during transmission. The IP datagram is forwarded in “as-is” condition to the Transmission Control Protocol (TCP) layer at the receiving end. The TCP then has to make a request for datagrams that are either missing or contain errors.

Among other capabilities, IP includes support for the fragmentation of larger packets into multiple smaller packets. When one computer uses IP to communicate with another, the instructions for how to put the fragments back together are contained within the IP Header. IP fragmentation is the process of breaking up a single IP packet into multiple packets of smaller size. This is a normal behavior on IP networks and is not in itself an indicator of attack. Therefore, inline security solutions conducting deep inspection must reassemble IP fragments before inspection can occur. If the programmers developing the product made a mistake (and developers make mistakes all the time) reassembling IP packets, an attacker may be able to evade detection by fragmenting the IP packets in any number of ways, such as sending them in reverse order, delaying the first fragment, or sending overlapping duplicate fragments with garbage payload.

TCP Segmentation

TCP is one of the main protocols that run atop of the IP. Where IP is stateless, TCP is stateful, meaning that it tracks what has been sent and received via the TCP/IP. Just as IP can be fragmented, so too can TCP. When one computer uses TCP/IP to communicate with another, the instructions for how to put the TCP segments back together are contained within the TCP Header. This is common within network traffic and is not itself an indicator of an attack. Therefore, inline security solutions conducting deep inspection must reassemble TCP streams before inspection can occur. If the programmers developing the product made a mistake reassembling TCP streams, an attacker may be able to evade detection by segmenting the TCP streams in any number of ways, such as sending them in reverse

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

[4] The results of resiliency testing are included in the exploit block rate calculations.

order, delaying the first segment, or sending overlapping duplicate segments with garbage payload. In addition, an attacker can combine evasion techniques both segmenting TCP and fragmenting IP.

HTTP Obfuscation & Compression

Web browsers request content from servers over HTTP using the ASCII character-set. HTTP encoding replaces unsafe non-ASCII characters with a "%" followed by two hexadecimal digits. Web servers and clients understand how to decode the request and responses. However, this mechanism can be abused to circumvent protection that is looking to match specific strings of characters. Sample methods include chunked encoding and header folding.

Chunked encoding allows the server to break a document into smaller chunks and transmit the chunks individually. The server needs only to specify the size of each chunk before it is transmitted and then indicate when the last chunk has been transmitted. Since chunked encoding intersperses arbitrary numbers (chunk sizes) with the elements of the original document, it can be used to greatly change the appearance of the content as observed "on the wire" during transmission. In addition, the server can choose to break the document into chunks at arbitrary points. This makes it difficult to reliably identify the original HTML content from the raw data on the network.

Per RFC 2616, the HTTP protocol allows the server to use several compression methods. These compression methods not only improve performance, but in many circumstances, they completely change the characteristic size and appearance of HTML documents. Small changes in the original document can greatly change the final appearance of the compressed document. This property of these algorithms could be used to obfuscate hostile content for the purpose of evading detection. The deflate compression method is a Lempel-Ziv coding (LZ77), specified in RFC 1951. The gzip compression method is specified in RFC 1952.

HTML Obfuscation[5]

HTML is a file type that a web server transmits via HTTP to a web browser, which the browser then renders for the user. So, whereas HTTP obfuscations evade detection by manipulating the transmission, HTML obfuscations are contained within the content itself. It is important that security solutions charged with protecting end systems correctly interpret HTML content and have semantic or syntactic understanding of the data they are analyzing. Otherwise, they could be vulnerable to evasions through the use of redundant, but equivalent, alternative representations of malicious content. For example, an attacker can encode HTML content using different UTF encoding. A security product that does not properly decode the content will miss the attack. This test suite uses malicious HTML content that is transferred from web server to web browser.

Protection Resiliency[6]

NSS defines resilience as a product's capability to continue providing protection for a vulnerability against a known exploit after various modifications have been made to the original exploit. Different variations of an exploit can be used to exploit a vulnerability. And many security vendors claim their solutions provide vulnerability-based protection that will block exploitation of vulnerabilities regardless of the specific exploit. A product that is able to defend against multiple exploit variations provides resilient protection.

Resilience is an important capability since exploit modifications often require little technical skill to implement. For example, a script-based drive-by exploit delivered in HTML may include a great deal of content that can be easily

[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.

modified, such as the names of variables and functions; the order in which they are declared; adding and removing whitespace; swapping payload; changing comments; and/or a combination of the techniques in the below table. Enterprises rely on network security products to provide protection for unpatched and/or vulnerable applications in their environments, picking off exploitation attempts “on-the-wire” for scale and efficiency. The goal of NSS' resilience testing is to evaluate the quality of a product's vulnerability protection signatures in order to determine whether they can adequately detect or match the essential exploitation elements of a vulnerability (in other words, the “trigger”), or can be easily bypassed by an adversary making simple changes to a readily available POC exploit.

Inline security products are largely dependent on pattern-matching signatures (e.g., “match if this string of bytes is seen within x bytes of this other string of bytes”) to identify malicious content after normalization of network streams. These products are unlikely to further process the content (e.g., decode, deobfuscate, or render the HTML and associated script) prior to inspection due to the impact it would have on throughput. Note that script obfuscations and resiliency are included in the security effectiveness score since these attacks are considered “complex evasions” (HTML/JavaScript/VBScript) and require real-time code analysis in order to determine whether a function is legitimate or obfuscating an attack. For details, please see Appendix A: Product Scorecard.

In each of the unique test cases, the baseline threat was blocked before the resiliency technique(s) was applied. Results for each test case cite specific resiliency modifications applied to the baseline and the outcome. To stress resilience capabilities and to reveal any issues, techniques were iterated and reordered. During execution, the results were observed on a victim target machine. For detailed results, see Appendix A: Product Scorecard.

Performance

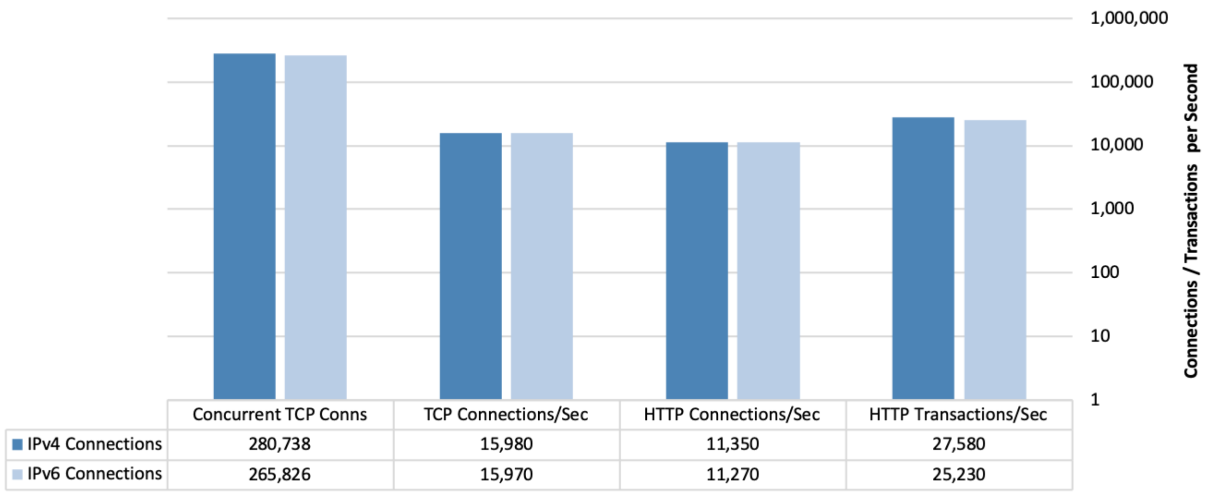
There is frequently a trade-off between security effectiveness and performance. Because of this trade-off, it is important to judge a product's security effectiveness within the context of its performance and vice versa. This ensures that new security protections do not adversely impact performance and that security shortcuts are not taken to maintain or improve performance. Performance is measured for both the IPv4 and the IPv6 network protocol.

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 is 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 contain valid payload and address data, and 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—are used:

- **Excessive concurrent TCP connections** – Latency within the NGIPS is causing an unacceptable increase in open connections.
- **Excessive concurrent HTTP connections** – Latency within the NGIPS 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 NGIPS is causing connections to time out.



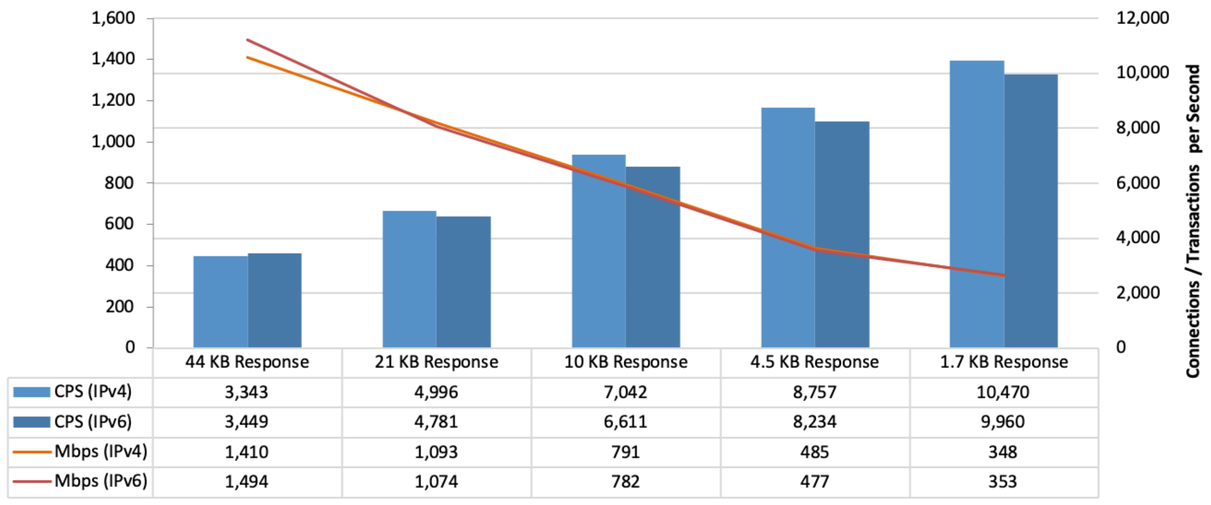
	Concurrent TCP Conns	TCP Connections/Sec	HTTP Connections/Sec	HTTP Transactions/Sec
■ IPv4 Connections	280,738	15,980	11,350	27,580
□ IPv6 Connections	265,826	15,970	11,270	25,230

Figure 8 – Concurrency and Connection Rates

HTTP Capacity

The aim of these tests is 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 is 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 possible, while ensuring absolute accuracy and repeatability.

Each transaction consists of a single HTTP GET request and there are no transaction delays; i.e., the web server responds immediately to all requests. All packets contain valid payload (a mix of binary and ASCII objects) and address data. This test provides an excellent representation of a live network (albeit one biased toward HTTP traffic) at various network loads.



	44 KB Response	21 KB Response	10 KB Response	4.5 KB Response	1.7 KB Response
▬ CPS (IPv4)	3,343	4,996	7,042	8,757	10,470
▬ CPS (IPv6)	3,449	4,781	6,611	8,234	9,960
— Mbps (IPv4)	1,410	1,093	791	485	348
— Mbps (IPv6)	1,494	1,074	782	477	353

Figure 9 – HTTP Connections per Second and Capacity

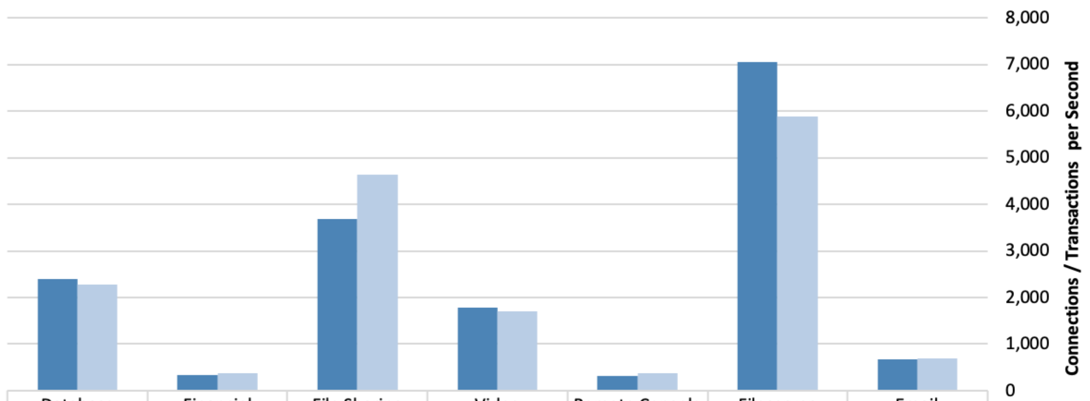
Application Average Response Time – HTTP

Application Average Response Time – HTTP (at 90% Maximum Load)	IPv4 Results	IPv6 Results
2,500 Connections per Second – 44-KB Response	6.46	7.19
5,000 Connections per Second – 21-KB Response	5.25	5.74
10,000 Connections per Second – 10-KB Response	5.73	6.04
20,000 Connections per Second – 4.5-KB Response	3.41	2.5
40,000 Connections per Second – 1.7-KB Response	4.85	4.33

Figure 10 – Average Application Response Time (Milliseconds)

Single Application Flows

This test measures the performance of the device with single application flows. For details about single application flow testing, please see Appendix A: Product Scorecard.



	Database	Financial	File Sharing	Video	Remote Console	Fileserver	Email
■ Mbps (IPv4)	2,401	327	3,693	1,785	324	7,054	678
□ Mbps (IPv6)	2,283	376	4,633	1,707	384	5,888	702

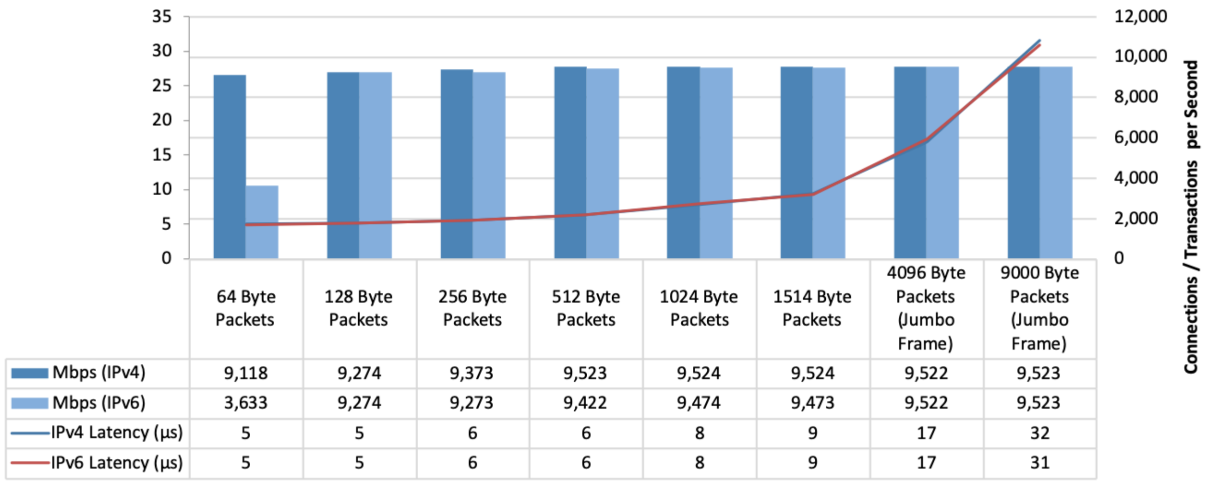
Figure 11 – Single Application Flows

Raw Packet Processing Performance (UDP Throughput)

This test uses UDP packets of varying sizes generated by test equipment. 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, is transmitted bidirectionally through each port pair of the device.

Each packet contains dummy data and is 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 are verified by network monitoring tools before each test begins. Multiple tests are run and averages are taken where necessary.

This traffic does not attempt to simulate a real-world network condition. No TCP sessions are created during this test, and there is very little for the state engine to do. The aim of this test is to determine the raw packet processing capability of each inline port pair of the device, and to determine the device's effectiveness at forwarding packets quickly, in order to provide the highest level of network performance with the least amount of latency.



	64 Byte Packets	128 Byte Packets	256 Byte Packets	512 Byte Packets	1024 Byte Packets	1514 Byte Packets	4096 Byte Packets (Jumbo Frame)	9000 Byte Packets (Jumbo Frame)
Mbps (IPv4)	9,118	9,274	9,373	9,523	9,524	9,524	9,522	9,523
Mbps (IPv6)	3,633	9,274	9,273	9,422	9,474	9,473	9,522	9,523
IPv4 Latency (μs)	5	5	6	6	8	9	17	32
IPv6 Latency (μs)	5	5	6	6	8	9	17	31

Figure 12 – Raw Packet Processing Performance (UDP Traffic)

Raw Packet Processing Performance (UDP Latency)

NGIPS 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 13 depicts UDP latency (in microseconds) as recorded during the UDP throughput tests at 90% of maximum load.

Latency – UDP	IPv4 Results	IPv6 Results
64-Byte Packets	5.01	5.02
128-Byte Packets	5.21	5.23
256-Byte Packets	5.57	5.34
512-Byte Packets	6.34	6.41
1,024-Byte Packets	7.90	7.87
1,514-Byte Packets	9.41	9.69
4,096-Byte Packets	16.94	16.80
9,000-Byte Packets	31.63	31.50

Figure 13 – UDP Latency in Microseconds

NSS-Tested Throughput

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

Product	NSS-Tested Throughput (IPV4)	NSS-Tested Throughput (IPv6)
Fortinet FortiGate-100F v6.0.2 build6215 (GA) (IPS engine 4.00045 and signature pack 14.00697)	3,084	3,047

Figure 14 – NSS-Tested Throughput (Mbps)

Stability and Reliability

Long-term stability is particularly important for an inline device, where failure can produce network outages. These tests verify the stability of the device along with its ability to maintain security effectiveness while under normal load and while passing malicious traffic. Products that cannot sustain legitimate traffic (or that crash) while under hostile attack will not pass. Stability and reliability was tested over IPv4 only.

The device is required to remain operational and stable throughout these tests and to block 100% of previously blocked traffic, raising an alert for each. If any non-allowed traffic passes successfully, caused either by the volume of traffic or by the device failing open for any reason, it will fail the test.

Stability and Reliability	Result
Blocking Under Extended Attack	PASS
Passing Legitimate Traffic Under Extended Attack	PASS
Power Fail Recovery	PASS
Power Redundancy	PASS
Power Fail Open (No Inspection)	*See Footnote*[7]
Persistence of Data	PASS

Figure 15 – Stability and Reliability Results

These tests also determine the behavior of the state engine under load. All NGIPS must choose whether to risk denying legitimate traffic or risk allowing malicious traffic once they run low on resources. A product will drop new connections when resources (such as state table memory) are low, or when traffic loads exceed its capacity. In theory, this means the NGIPS will block legitimate traffic but maintain state on existing connections (and prevent attack leakage).

These tests also determine the behavior of the device during a complete loss of power. The expected behavior is that when power is restored, the device will return to normal operation, with configuration and log data intact. The device should not require any manual intervention to return to an operational state.

Normally, a device will block all traffic during a loss of power. However, the device may include optional hardware that will allow all traffic to pass uninspected until the device is again operational. The Power Fail Open test verifies this optional hardware feature.

[7] It was found that the Fortinet FortiGate-100F did not possess fail-open capabilities. Fortinet did not provide optional hardware to support the test case.

Appendix A: Product Scorecard

Security Effectiveness	
Intrusion Prevention Policies	
False Positive Testing	PASS
Exploit Block Rate	**99.91%**
Exploit Library Block Rate	99.89%
Live Exploit Block Rate	100.00%
Script Obfuscation Block Rate	100.00%
Evasions and Attack Leakage	
Resistance to Evasion	
TCP Split Handshake	PASS
HTTP Evasions	
Content delivered unencrypted over TCP port 443	PASS
HTTP/0.9 response (no response headers)	PASS
Declared HTTP/0.9 response; but includes response headers; chunking declared but served without chunking	PASS
Declared HTTP/0.9 response with gzip compression declared; served compressed; invalid content-length	PASS
Declared HTTP/0.9 response; but includes response headers; chunking declared; chunk with some data in chunk-extension field	PASS
HTTP/1.1 response compressed with gzip; invalid content-length	PASS
HTTP/1.1 response declaring gzip followed by junk string; invalid content-length; served uncompressed	PASS
HTTP/1.1 response declaring gzip with "Transfer-Encoding: gzip" header; invalid content-length; served uncompressed	PASS
HTTP/1.1 response declaring gzip with "Content-Encoding: gzip" header followed by a comma; invalid content-length; served uncompressed	PASS
HTTP/1.1 response compressed with deflate; invalid content-length	PASS
HTTP/1.1 response declaring deflate followed by junk string; invalid content-length; served uncompressed	PASS
HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30')	PASS
HTTP/1.1 chunked response with chunk sizes followed by end of transmission (hex '04')	PASS
HTTP/1.1 chunked response with chunk sizes followed by end of transmission block (hex '17')	PASS
HTTP/1.1 chunked response with chunk sizes followed by file separator (hex '1c')	PASS
HTTP/1.1 chunked response with chunk sizes followed by comma (hex '2c')	PASS
HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a $ (hex '24')	PASS
HTTP/1.1 chunked response with final chunk size of '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000' (rather than '0')	PASS
HTTP/1.1 response with line folded transfer-encoding header declaring chunking ('Transfer-Encoding: ' followed by CRLF (hex '0d 0a') followed by 'chunked' followed by CRLF (hex '0d 0a'); served without chunking	PASS
HTTP/1.1 response with transfer-encoding header declaring chunking with lots of whitespace ('Transfer-Encoding:' followed by 8000 spaces (hex '20' * 8000) followed by 'chunked' followed by CRLF (hex '0d 0a'); served chunked	PASS
HTTP/1.0 response declaring chunking; served without chunking	PASS
HTTP/1.0 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\n; chunked	PASS
HTTP/1.1 response with header end \r\n\010\r\n\r\n; chunked	PASS
HTTP/1.1 response with header end \n\r\r\n; chunked	PASS
HTTP/1.1 response with header end \n\006\011\n\n; chunked	PASS
HTTP/1.1 response with header end \n\033\n\003\n\n; chunked	PASS
HTTP/1.1 response with 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 followed by a comma; 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
No status line; chunking indicated; chunk with some data in chunk-extension field	PASS
Double Transfer-Encoding: first empty; last chunked. Served with invalid content-length; chunk with some data in chunk-extension field	PASS
HTTP/1.1 response with "Transfer-Encoding: chunked" header followed by a comma; chunk with some data in chunk-extension field	PASS
HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes followed by end of transmission (hex '04'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes followed by end of transmission block (hex '17'); compressed with gzip	PASS

HTTP/1.1 chunked response with chunk sizes followed by file separator (hex '1c'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes followed by comma (hex '2c'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a $ (hex '24'); compressed with gzip	PASS
HTTP/1.1 chunked response with chunk sizes preceded by multiple zeros (hex '30'); compressed with deflate	PASS
HTTP/1.1 chunked response with chunk sizes followed by end of transmission (hex '04'); compressed with deflate	PASS
HTTP/1.1 chunked response with chunk sizes followed by end of transmission block (hex '17'); compressed with deflate	PASS
HTTP/1.1 chunked response with chunk sizes followed by file separator (hex '1c'); compressed with deflate	PASS
HTTP/1.1 chunked response with chunk sizes followed by comma (hex '2c'); compressed with deflate	PASS
HTTP/1.1 chunked response with chunk sizes followed by a space (hex '20') then a $ (hex '24'); compressed with deflate	PASS
Network Evasions	
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
small IP fragments; overlapping duplicate fragments with garbage payloads; chunked	PASS
small IP fragments in reverse order; chunked	PASS
small IP fragments in random order; chunked	PASS
small IP fragments; delay first fragment; chunked	PASS
small IP fragments in reverse order; delay last fragment; chunked	PASS
small IP fragments; interleave chaff after (invalid IP options); chunked	PASS
small IP fragments in random order; interleave chaff sandwich (invalid IP options); chunked	PASS
small IP fragments in random order; interleave chaff sandwich (invalid IP options); delay random fragment; chunked	PASS
small IP fragments; interleave chaff before (invalid IP options); DSCP value 16; chunked	PASS
small IP fragments in random order; interleave chaff after (invalid IP options); delay random fragment; DSCP value 34; chunked	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
small IP fragments; overlapping duplicate fragments with garbage payloads; chunked; compressed with gzip	PASS
small IP fragments in reverse order; chunked; compressed with gzip	PASS
small IP fragments in random order; chunked; compressed with gzip	PASS
small IP fragments; delay first fragment; chunked; compressed with gzip	PASS

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

small TCP segments; interleave chaff (invalid IP options; reserved flags set); chunked	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points before first address); chunked	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points past last address); chunked	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points to middle of first address); chunked	PASS
small TCP segments; interleave chaff (invalid IP options; more than two loose source route options); chunked	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points before first address); chunked	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points past last address)); chunked	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points to middle of first address); chunked	PASS
small TCP segments; interleave chaff (invalid IP options; more than two strict source route options); chunked	PASS
small TCP segments; interleave chaff (invalid IP options; invalid timestamp option overflow_flag field); chunked	PASS
small TCP segments; interleave chaff (invalid IP options; invalid timestamp option pointer points past last address); chunked	PASS
small TCP segments; small IP fragments; chunked; compressed with gzip	PASS
small TCP segments; small IP fragments in reverse order; chunked; compressed with gzip	PASS
small TCP segments in random order; small IP fragments; chunked; compressed with gzip	PASS
small TCP segments; small IP fragments in random order; chunked; compressed with gzip	PASS
small TCP segments in random order; small IP fragments in reverse order; chunked; compressed with gzip	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); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IP fragments; interleave chaff before (invalid IP options); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IP fragments; interleave chaff sandwich (invalid IP options); delay last fragment; chunked; compressed with gzip	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; chunked; compressed with gzip	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; chunked; compressed with gzip	PASS
small overlapping TCP segments; small fragments; chunked; compressed with gzip	PASS
small overlapping TCP segments; delay last segment; small fragments; delay last fragment; chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; invalid length); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; reserved flags set); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points before first address); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points past last address); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points to middle of first address); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; more than two loose source route options); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points before first address); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points past last address)); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points to middle of first address); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; more than two strict source route options); chunked; compressed with gzip	PASS

small TCP segments; interleave chaff (invalid IP options; invalid timestamp option overflow_flag field); chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid IP options; invalid timestamp option pointer points past last address); chunked; compressed with gzip	PASS
small TCP segments; small IP fragments; chunked; compressed with deflate	PASS
small TCP segments; small IP fragments in reverse order; chunked; compressed with deflate	PASS
small TCP segments in random order; small IP fragments; chunked; compressed with deflate	PASS
small TCP segments; small IP fragments in random order; chunked; compressed with deflate	PASS
small TCP segments in random order; small IP fragments in reverse order; chunked; compressed with deflate	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); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IP fragments; interleave chaff before (invalid IP options); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IP fragments; interleave chaff sandwich (invalid IP options); delay last fragment; chunked; compressed with deflate	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; chunked; compressed with deflate	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; chunked; compressed with deflate	PASS
small overlapping TCP segments; small fragments; chunked; compressed with deflate	PASS
small overlapping TCP segments; delay last segment; small fragments; delay last fragment; chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; invalid length); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; reserved flags set); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points before first address); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points past last address); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points to middle of first address); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; more than two loose source route options); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points before first address); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points past last address)); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points to middle of first address); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; more than two strict source route options); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; invalid timestamp option overflow_flag field); chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid IP options; invalid timestamp option pointer points past last address); chunked; compressed with deflate	PASS
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 IPv6 fragments in reverse order; delay last fragment; chunked	PASS
small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay random fragment; chunked	PASS
small IPv6 fragments in random order; delay first fragment; chunked	PASS
small IPv6 fragments in random order; delay last fragment; chunked	PASS
small IPv6 fragments in random order; delay random fragment; chunked	PASS
small IPv6 fragments; chunked; compressed with gzip	PASS
small IPv6 fragments in reverse order; chunked; compressed with gzip	PASS
small IPv6 fragments in random order; chunked; compressed with gzip	PASS
small IPv6 fragments; delay first fragment; chunked; compressed with gzip	PASS
small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay first fragment; chunked; compressed with gzip	PASS
small IPv6 fragments in reverse order; delay last fragment; chunked; compressed with gzip	PASS
small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay random fragment; chunked; compressed with gzip	PASS
small IPv6 fragments in random order; delay first fragment; chunked; compressed with gzip	PASS
small IPv6 fragments in random order; delay last fragment; chunked; compressed with gzip	PASS
small IPv6 fragments in random order; delay random fragment; chunked; compressed with gzip	PASS
small IPv6 fragments; chunked; compressed with deflate	PASS
small IPv6 fragments in reverse order; chunked; compressed with deflate	PASS
small IPv6 fragments in random order; chunked; compressed with deflate	PASS
small IPv6 fragments; delay first fragment; chunked; compressed with deflate	PASS
small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay first fragment; chunked; compressed with deflate	PASS
small IPv6 fragments in reverse order; delay last fragment; chunked; compressed with deflate	PASS
small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay random fragment; chunked; compressed with deflate	PASS
small IPv6 fragments in random order; delay first fragment; chunked; compressed with deflate	PASS
small IPv6 fragments in random order; delay last fragment; chunked; compressed with deflate	PASS
small IPv6 fragments in random order; delay random fragment; chunked; compressed with deflate	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 TCP segments; interleave chaff (invalid TCP checksums); delay first segment; chunked	PASS
small TCP segments in random order; interleave chaff after (older PAWS timestamps); delay last segment; chunked	PASS
small TCP segments in random order; interleave chaff before (out-of-window sequence numbers); TCP MSS option; chunked	PASS
small TCP segments in random order; interleave chaff sandwich (requests to resynch sequence numbers mid-stream); TCP window scale option; chunked	PASS
small TCP segments in random order; interleave chaff before (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment; chunked	PASS
small overlapping TCP segments; chunked	PASS
small TCP segments; chunked; compressed with gzip	PASS
small TCP segments in reverse order; chunked; compressed with gzip	PASS
small TCP segments in random order; chunked; compressed with gzip	PASS
small TCP segments; delay first segment; chunked; compressed with gzip	PASS
small TCP segments in reverse order; delay last segment; chunked; compressed with gzip	PASS
small TCP segments; interleave chaff (invalid TCP checksums); delay first segment; chunked; compressed with gzip	PASS
small TCP segments in random order; interleave chaff after (older PAWS timestamps); delay last segment; chunked; compressed with gzip	PASS
small TCP segments in random order; interleave chaff before (out-of-window sequence numbers); TCP MSS option; chunked; compressed with gzip	PASS
small TCP segments in random order; interleave chaff sandwich (requests to resynch sequence numbers mid-stream); TCP window scale option; chunked; compressed with gzip	PASS
small TCP segments in random order; interleave chaff before (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment; chunked; compressed with gzip	PASS
small overlapping TCP segments; chunked; compressed with gzip	PASS
small TCP segments; chunked; compressed with deflate	PASS
small TCP segments in reverse order; chunked; compressed with deflate	PASS
small TCP segments in random order; chunked; compressed with deflate	PASS
small TCP segments; delay first segment; chunked; compressed with deflate	PASS
small TCP segments in reverse order; delay last segment; chunked; compressed with deflate	PASS
small TCP segments; interleave chaff (invalid TCP checksums); delay first segment; chunked; compressed with deflate	PASS
small TCP segments in random order; interleave chaff after (older PAWS timestamps); delay last segment; chunked; compressed with deflate	PASS
small TCP segments in random order; interleave chaff before (out-of-window sequence numbers); TCP MSS option; chunked; compressed with deflate	PASS
small TCP segments in random order; interleave chaff sandwich (requests to resynch sequence numbers mid-stream); TCP window scale option; chunked; compressed with deflate	PASS
small TCP segments in random order; interleave chaff before (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment; chunked; compressed with deflate	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 TCP segments in random order; interleave chaff before (invalid TCP checksums); small IPv6 fragments in reverse order; chunked	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IPv6 fragments; chunked	PASS
small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IPv6 fragments; delay last fragment; chunked	PASS
small TCP segments in random order; interleave chaff sandwich (out-of-window sequence numbers); small IPv6 fragments in random order; delay random fragment; chunked	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; 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 TCP segments in random order; interleave chaff before (invalid TCP checksums); small IPv6 fragments in reverse order; chunked; compressed with gzip	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IPv6 fragments; chunked; compressed with gzip	PASS
small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IPv6 fragments; delay last fragment; chunked; compressed with gzip	PASS
small TCP segments in random order; interleave chaff sandwich (out-of-window sequence numbers); small IPv6 fragments in random order; delay random fragment; chunked; compressed with gzip	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; 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 TCP segments in random order; interleave chaff before (invalid TCP checksums); small IPv6 fragments in reverse order; chunked; compressed with deflate	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IPv6 fragments; chunked; compressed with deflate	PASS
small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IPv6 fragments; delay last fragment; chunked; compressed with deflate	PASS
small TCP segments in random order; interleave chaff sandwich (out-of-window sequence numbers); small IPv6 fragments in random order; delay random fragment; chunked; compressed with deflate	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; 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[8]	
js-binary-obfuscation*	PASS
babel-minify*	PASS
closure*	PASS
code-protect*	PASS
confusion*	PASS
jfogs*	PASS
jfogs-reverse*	PASS
jjencode*	PASS
jsbeautifier*	PASS
jsmin*	PASS
js-obfuscator*	PASS
qzx-obfuscator*	PASS
js-binary-obfuscation; chunked*	PASS
babel-minify; chunked*	PASS
closure; chunked*	PASS
code-protect; chunked*	PASS
confusion; chunked*	PASS
jfogs; chunked*	PASS
jfogs-reverse; chunked*	PASS
jjencode; chunked*	PASS
jsbeautifier; chunked*	PASS
jsmin; chunked*	PASS
js-obfuscator; chunked*	PASS
qzx-obfuscator; chunked*	PASS
chunked and gzip compressed js-binary-obfuscation*	PASS

[8] Script obfuscations annotated with an* are included in the exploit block rate calculations.

chunked and gzip compressed babel-minify*	PASS
chunked and gzip compressed closure*	PASS
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*	PASS
chunked and gzip compressed jsbeautifier*	PASS
chunked and gzip compressed jsmin*	PASS
chunked and gzip compressed js-obfuscator*	PASS
chunked and gzip compressed qzx-obfuscator*	PASS
chunked and deflate compressed js-binary-obfuscation*	PASS
chunked and deflate compressed babel-minify*	PASS
chunked and deflate compressed closure*	PASS
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*	PASS
chunked and deflate compressed jsbeautifier*	PASS
chunked and deflate compressed jsmin*	PASS
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 and gzip compressed	PASS
UTF-8 encoding; chunked and gzip compressed	PASS
UTF-8 encoding with BOM; chunked and gzip compressed	PASS
UTF-16 encoding with BOM; chunked and gzip compressed	PASS
UTF-8 encoding; no http or html declarations; chunked and gzip compressed	PASS
UTF-8 encoding with BOM; no http or html declarations; chunked and gzip compressed	PASS
UTF-16 encoding with BOM; no http or html declarations; chunked and gzip compressed	PASS
UTF-16-LE encoding without BOM; chunked and gzip compressed	PASS
UTF-16-BE encoding without BOM; chunked and gzip compressed	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-7 encoding; chunked and deflate compressed	PASS
UTF-8 encoding; chunked and deflate compressed	PASS
UTF-8 encoding with BOM; chunked and deflate compressed	PASS
UTF-16 encoding with BOM; chunked and deflate compressed	PASS
UTF-8 encoding; no http or html declarations; chunked and deflate compressed	PASS
UTF-8 encoding with BOM; no http or html declarations; chunked and deflate compressed	PASS
UTF-16 encoding with BOM; no http or html declarations; chunked and deflate compressed	PASS
UTF-16-LE encoding without BOM; chunked and deflate compressed	PASS
UTF-16-BE encoding without BOM; chunked and deflate 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
UTF-7 encoding; chunked and deflate compressed	PASS
EICAR string included at top of HTML; comments removed	PASS
EICAR string included at top of HTML; comments removed; chunked	PASS
EICAR string included at top of HTML; comments removed; chunked and gzip compressed	PASS
EICAR string included at top of HTML; comments removed; chunked and deflate compressed	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
Hex encoded script decoded using JavaScript unescape; chunked and gzip compressed*	PASS
Unicode encoded script decoded using JavaScript unescape; chunked and gzip compressed*	PASS
Hex encoded script as variable decoded using JavaScript unescape; chunked and gzip compressed*	PASS
Unicode encoded script as variable decoded using JavaScript unescape; chunked and gzip compressed*	PASS
Hex encoded script decoded using JavaScript unescape; chunked and deflate compressed*	PASS
Unicode encoded script decoded using JavaScript unescape; chunked and deflate compressed*	PASS
Hex encoded script as variable decoded using JavaScript unescape; chunked and deflate compressed*	PASS

Unicode encoded script as variable decoded using JavaScript unescape; chunked and deflate compressed*	PASS
padded with <=5MB	PASS
padded with <=25MB	PASS
padded with >25MB	PASS
padded with <=5MB	PASS
padded with <=20MB	PASS
padded with >20MB	PASS
padded with <=5MB; chunked	PASS
padded with <=25MB; chunked	PASS
padded with >25MB; chunked	PASS
padded with <=5MB; chunked	PASS
padded with <=20MB; chunked	PASS
padded with >20MB; 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 <=20MB; chunked and compressed with gzip	PASS
padded with >20MB; 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 <=20MB; chunked and compressed with deflate	PASS
padded with >20MB; chunked and compressed with deflate	PASS
Evasion Resilience[9]	
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"	PASS
VBScript interspersed randomly with null bytes; content="IE=10" replaced with content="IE=EmulateIE8"; chunked	PASS

[9] The results of resiliency testing are included in the exploit block rate calculations.

VBScript interspersed randomly with null bytes; content="IE=10" replaced with content="IE=EmulateIE8"; chunked and gzip compressed	PASS
VBScript interspersed randomly with null bytes; content="IE=10" replaced with content="IE=EmulateIE8"; chunked and deflate compressed	PASS
Veil Ordnance generated bind shell stager	PASS
Veil Ordnance generated bind shell stager; chunked	PASS
Veil Ordnance generated bind shell stager; chunked and gzip compressed	PASS
Veil Ordnance generated bind shell stager; chunked and deflate compressed	PASS
msfvenom generated bind shell stager	PASS
msfvenom generated bind shell stager; chunked	PASS
msfvenom generated bind shell stager; chunked and gzip compressed	PASS
msfvenom generated bind shell stager; chunked and deflate compressed	PASS
msfvenom generated bind shell stager; shikata_ga_nai encoded 10x	PASS
msfvenom generated bind shell stager; shikata_ga_nai encoded 10x; chunked	PASS
msfvenom generated bind shell stager; shikata_ga_nai encoded 10x; chunked and gzip compressed	PASS
msfvenom generated bind shell stager; shikata_ga_nai encoded 10x; chunked and deflate compressed	PASS
msfvenom generated bind shell stager; call4_dword_xor encoded 10x	PASS
msfvenom generated bind shell stager; call4_dword_xor encoded 10x; chunked	PASS
msfvenom generated bind shell stager; call4_dword_xor encoded 10x; chunked and gzip compressed	PASS
msfvenom generated bind shell stager; call4_dword_xor encoded 10x; chunked and deflate compressed	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()/Clng	PASS
Some script commands/strings converted to series of chr()	PASS
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	PASS
change chr() and chrw() to chrb(); Some script commands/strings converted to series of chr()/Clng; change all chr() to chrw() and vice versa where possible	PASS
combination of res-ord-501; res-wsp-501	PASS
combination of res-ren-501; res-wsp-501	PASS
combination of res-ren-501; res-ord-501	PASS
combination of res-ren-501; res-ord-501; res-wsp-501	PASS
combination of res-ren-501; res-ord-501; res-wsp-501; res-mth-503	PASS
combination of res-ren-501; res-ord-501; res-wsp-501; res-mth-503; res-spl-501	PASS
combination of res-ren-501; res-ord-501; res-mth-503; res-wsp-501; res-pay-501	PASS

combination of res-ren-501; res-ord-501; res-wsp-501; res-mth-503; res-spl-501; res-chr-506	PASS
combination of res-ren-501; res-ord-501; res-wsp-501; res-mth-503; res-spl-501; res-chr-505	PASS
combination of res-ren-501; res-ord-501; res-wsp-501; res-mth-503; res-spl-501; res-pay-501	PASS
combination of res-ren-501; res-ord-501; res-wsp-501; res-mth-503; res-spl-501; res-chr-506; res-pay-501	PASS
combination of res-ren-501; res-ord-501; res-wsp-501; res-mth-503; res-spl-501; res-chr-505; res-pay-501	PASS
combination of res-nb-501; res-ren-501; res-ord-501; res-wsp-501; res-spl-501; res-chr-506; res-pay-501	PASS
combination of res-nb-501; res-ren-501; res-ord-501; res-wsp-501; res-spl-501; res-chr-505; res-pay-501	PASS
combination of res-nb-501; res-ren-501; res-ord-501; res-wsp-501; res-spl-501; res-chr-506; res-pay-501; chunked	PASS
combination of res-nb-501; res-ren-501; res-ord-501; res-wsp-501; res-spl-501; res-chr-505; res-pay-501; chunked	PASS
combination of res-nb-501; res-ren-501; res-ord-501; res-wsp-501; res-spl-501; res-chr-506; res-pay-501; chunked and gzip compressed	PASS
combination of res-nb-501; res-ren-501; res-ord-501; res-wsp-501; res-spl-501; res-chr-505; res-pay-501; chunked and gzip compressed	PASS
combination of res-nb-501; res-ren-501; res-ord-501; res-wsp-501; res-spl-501; res-chr-506; res-pay-501; chunked and deflate compressed	PASS
combination of res-nb-501; res-ren-501; res-ord-501; res-wsp-501; res-spl-501; res-chr-505; res-pay-501; chunked and deflate compressed	PASS
Evasions using non-standard TCP ports	PASS
Evasion Combinations	
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 '00000000000000000000000000000000000000000000000000000000000000000000000000000000' (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
Performance – IPv4	
Throughput – UDP (Included in weighting for NSS-Tested Throughput)	
64-Byte Packets	9,118
128-Byte Packets	9,274
256-Byte Packets	9,373
512-Byte Packets	9,523
1,024-Byte Packets	9,524
1,514-Byte Packets	9,524
4,096-Byte Packets	9,522
9,000-Byte Packets	9,523
Latency – UDP	
64-Byte Packets	5.01
128-Byte Packets	5.21
256-Byte Packets	5.57
512-Byte Packets	6.34
1,024-Byte Packets	7.90
1,514-Byte Packets	9.41
4,096-Byte Packets	16.94
9,000-Byte Packets	31.63
Maximum Capacity	
Theoretical Max. Concurrent TCP Connections	280,738
Max TCP Connections/Second	15,980
Max HTTP Connections/Second	11,350
Max HTTP Transactions/Second	27,580
HTTP Response (Included in weighting for NSS-Tested Throughput)	
2,500 Connections Per Second – 44-KB Response	3,343
5,000 Connections Per Second – 21-KB Response	4,996
10,000 Connections Per Second – 10-KB Response	7,042
20,000 Connections Per Second – 4.5-KB Response	8,757
40,000 Connections Per Second – 1.7-KB Response	10,470
Application Average Response Time - HTTP (at 90% Max Load)	

2,500 Connections Per Second – 44-KB Response	6.46
5,000 Connections Per Second – 21-KB Response	5.25
10,000 Connections Per Second – 10-KB Response	5.73
20,000 Connections Per Second – 4.5-KB Response	3.41
40,000 Connections Per Second – 1.7-KB Response	4.85
Single Application Throughput (Included in weighting for NSS-Tested Throughput)	
Database	2,401
Financial	327
File Sharing	3,693
Video	1,785
Remote Console	324
File Server	7,054
Email	678
Performance – IPv6	
Throughput – UDP (Included in weighting for NSS-Tested Throughput)	
64-Byte Packets	3,633
128-Byte Packets	9,274
256-Byte Packets	9,273
512-Byte Packets	9,422
1,024-Byte Packets	9,474
1,514-Byte Packets	9,473
4,096-Byte Packets	9,522
9,000-Byte Packets	9,523
Latency – UDP	
64-Byte Packets	5.02
128-Byte Packets	5.23
256-Byte Packets	5.34
512-Byte Packets	6.41
1,024-Byte Packets	7.87
1,514-Byte Packets	9.69
4,096-Byte Packets	16.80
9,000-Byte Packets	31.50
Maximum Capacity	
Theoretical Max. Concurrent TCP Connections	265,826
Max TCP Connections/Second	15,970
Max HTTP Connections/Second	11,270
Max HTTP Transactions/Second	25,230
HTTP Response (Included in weighting for NSS-Tested Throughput)	
2,500 Connections Per Second – 44-KB Response	3,139
5,000 Connections Per Second – 21-KB Response	4,781
10,000 Connections Per Second – 10-KB Response	6,611
20,000 Connections Per Second – 4.5-KB Response	8,234
40,000 Connections Per Second – 1.7-KB Response	9,960

Application Average Response Time - HTTP (at 90% Max Load)	
2,500 Connections Per Second – 44-KB Response	7.19
5,000 Connections Per Second – 21-KB Response	5,74
10,000 Connections Per Second – 10-KB Response	6.04
20,000 Connections Per Second – 4.5-KB Response	2.50
40,000 Connections Per Second – 1.7-KB Response	4.33
Single Application Throughput (Included in weighting for NSS-Tested Throughput)	
Database	2,283
Financial	376
File Sharing	4,633
Video	1,707
Remote Console	384
File Server	5,888
Email	702
Stability & Reliability	
Blocking Under Extended Attack	PASS
Passing Legitimate Traffic Under Extended Attack	PASS
Power PASS Recovery	PASS
Power Redundancy	PASS
Power PASS Open (No Inspection)	*See Footnote*[10]
Persistence of Data	PASS
Total Cost of Ownership	
Ease of Use	
Initial Setup (Hours)	8
Time Required for Upkeep (Hours per Year)	Contact NSS
Time Required to Tune (Hours per Year)	Contact NSS
Expected Costs	
Initial Purchase (hardware as tested)	$2,100
Installation Labor Cost (@$75/hr)	$600
Annual Cost of Maintenance & Support (hardware/software)	$945
Annual Cost of Updates (IPS/AV/etc.)	0
Initial Purchase (centralized management system)	Contact NSS
Annual Cost of Maintenance & Support (centralized management system)	Contact NSS
Management Labor Cost (per Year @$75/hr)	Contact NSS
Tuning Labor Cost (per Year @$75/hr)	Contact NSS
Total Cost of Ownership	
Year 1	$3,045
Year 2	$945

[10] It was found that the Fortinet FortiGate-100F did not possess fail-open capabilities. Fortinet did not provide optional hardware to support the test case.

Year 3	$945
3 Year Total Cost of Ownership	$4,935

Figure 16 – Detailed Scorecard

Test Methodology

Next Generation Intrusion Prevention System Test Methodology v5.0

A copy of the Test Methodology is available at www.nsslabs.com.

Contact Information

NSS Labs, Inc.
3711 South MoPac Expressway
Building 1, Suite 400
Austin, TX 78746-8022
USA
info@nsslabs.com
www.nsslabs.com