DATA CENTER SECURITY GATEWAY TEST REPORT

Palo Alto Networks PA-5250 9.0.3-h2

November 12, 2019

Author – Keith Bormann

Palo Alto Networks PA-5250 9.0.3-h2

Summary

In Q4 2019, NSS Labs performed an independent test of the Palo Alto Networks PA-5250 9.0.3-h2. This report focuses on the main differentiators for data center security gateway (DCSG) products: security, cost, and performance/functionality.

Security

The ideal DCSG delivers a high block rate for exploits and evasions while at the same time providing the expected stability and reliability.

Security Effectiveness	
Exploit Block Rate	99.45%
Evasions Blocked	126/126
Stability & Reliability	PASS
False Positives	PASS

Performance/Functionality

The ideal DCSG provides high performance while at the same time offering a high level of security.

Performance	
Transactional use case	9,467 Mbps
Multimedia use case	16,135 Mbps
Corporate use case	12,842 Mbps

Maximum Capacity	CPS
Theoretical Maximum Concurrent TCP Connections	8,386,560
Maximum TCP Connections/Second	176,900
Maximum HTTP Connections/Second	102,100
Maximum HTTP Transactions/Second	135,000

HTTP Capacity	CPS
2,500 Connections per Second – 44-KB Response	75,000
5,000 Connections per Second – 21-KB Response	98,850
10,000 Connections per Second – 10-KB Response	98,940
20,000 Connections per Second – 4.5-KB Response	100,900
40,000 Connections per Second – 1.7-KB Response	106,700

Cost

The ideal DCSG is scalable, delivers continuous uptime, and has low maintenance and support costs.

Total Cost of Ownership (TCO)	
3-Year TCO (US$)	$140,400

The product was subjected to thorough testing based on the Data Center Network Security (DCNS) Test Methodology v3.1 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.

Table of Contents

Table of Figures

Security Effectiveness

This section verifies that the device can enforce the security policy effectively.

NSS research indicates that DCSG devices are typically deployed to protect data center assets, and most enterprises will tune intrusion prevention system (IPS) modules within their DCSG. Therefore, during NSS testing, DCSG products are configured with a tuned policy setting in order to provide readers with relevant security effectiveness and performance dimensions based on their expected usage.

NSS Exploit Library

NSS' security effectiveness testing leverages the deep expertise of our engineers who utilize multiple commercial, open-source, and proprietary tools as appropriate. With more than 2,300 exploits, this is the industry's most comprehensive test to date.

Product	Total Number of Threats Run	Total Number of Threats Blocked	Block Percentage
Palo Alto Networks PA-5250 9.0.3-h2	2,363	2,350	99.45%

Figure 1 – Number of Threats Blocked (%)

Coverage by Date

Figure 2 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.

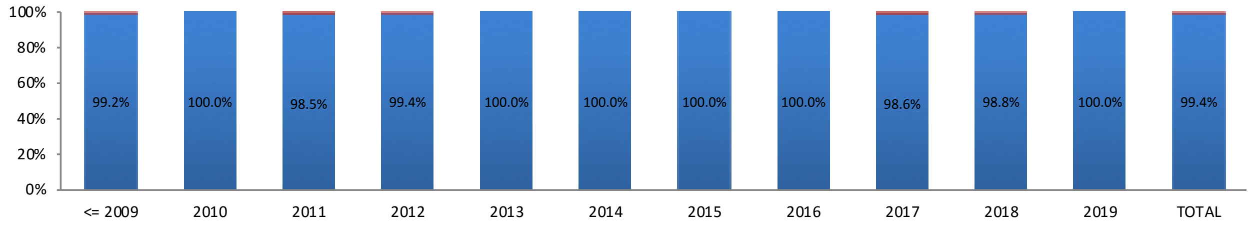


Figure 2 – Product Coverage by Date

Coverage by Target Vendor

Exploits within the *NSS Exploit Library* target a wide range of protocols and applications. Figure 3 depicts the coverage offered for some of the top vendors targeted in this test. Clients can contact NSS for more information.

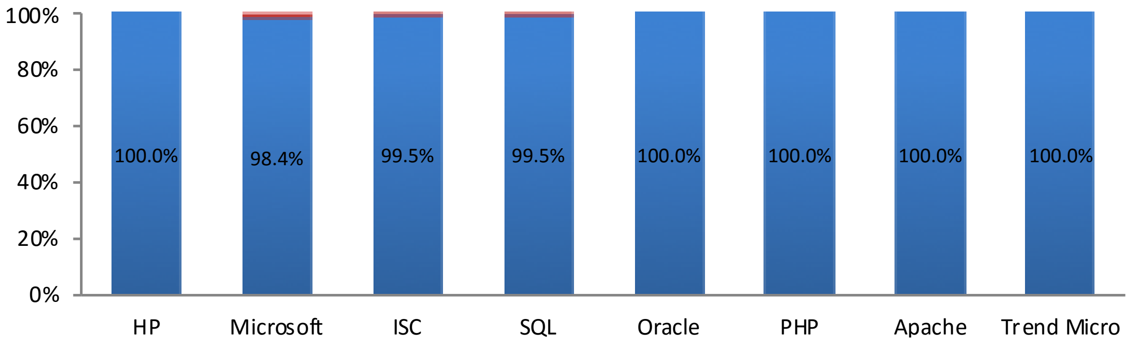


Figure 3 – Product Coverage by Target Vendor

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 IP packet fragmentation, TCP stream segmentation and RPC evasions), the less effective the device. For example, it is better to miss all techniques in one evasion category, such as RPC evasions, 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 (i.e., URL 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, access to the victim machine became available through a shell and the victim machine was compromised. Victim machines in the test harness did not have endpoints installed.

Devices were tested against 132 evasions; 126 of which were included in the *Evasions* section. The remaining six were resiliency evasions that were included in the *Block Rate*.

Figure 4 provides the results of the evasion tests for the PA-5250. The device blocked all 126 of the evasions used to calculate its *Evasions* score. For further detail, please reference Appendix A.

Test	Results
IP Packet Fragmentation	PASS
TCP Segmentation	PASS
Resiliency	See footnote[1]
○ Payload	PASS
○ Trigger	PASS
○ White Space	PASS
RPC Fragmentation	PASS
URL Obfuscation	PASS
FTP Evasion	PASS
Telnet Evasions	PASS

Figure 4 – Resistance to Evasion Results

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

IP Packet Fragmentation

IP Packet Fragmentation	Results
small IP fragments; overlapping duplicate fragments with garbage payloads (server-side exploit)	PASS
small overlapping IP fragments in reverse order (server-side exploit)	PASS
small overlapping IP fragments in random order (server-side exploit)	PASS
small IP fragments; delay first fragment (server-side exploit)	PASS
small IP fragments in reverse order; delay last fragment (server-side exploit)	PASS
small IP fragments; interleave chaff after (invalid IP options) (server-side exploit)	PASS
small IP fragments in random order; interleave chaff sandwich (invalid IP options) (server-side exploit)	PASS
small overlapping IP fragments in random order; interleave chaff sandwich (invalid IP options); delay random fragment (server-side exploit)	PASS
small IP fragments; interleave chaff before (invalid IP options); DSCP value 16 (server-side exploit)	PASS
small IP fragments in random order; interleave chaff after (invalid IP options); delay random fragment; DSCP value 34 (server-side exploit)	PASS
IPv4 fragmentation with an overlapping atomic fragment with good data inserted in-between the fragments with junk data (server-side exploit)	PASS
IPv4 fragmentation with an overlapping atomic fragment with junk data inserted in-between the fragments with good data (server-side exploit)	PASS
small IPv6 fragments (server-side exploit)	PASS
small IPv6 fragments in reverse order (server-side exploit)	PASS
small IPv6 fragments in random order (server-side exploit)	PASS
small IPv6 fragments; delay first fragment (server-side exploit)	PASS
small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay first fragment (server-side exploit)	PASS
small IPv6 fragments in reverse order; delay last fragment (server-side exploit)	PASS
small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay random fragment (server-side exploit)	PASS
small IPv6 fragments in random order; delay first fragment (server-side exploit)	PASS
small IPv6 fragments in random order; delay last fragment (server-side exploit)	PASS
small IPv6 fragments in random order; delay random fragment (server-side exploit)	PASS

Figure 5 – IP Fragmentation Results

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.

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.

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 missing or that 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 reassembling IP packets (and developers make mistakes all the time), 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 Segmentation	Results
small TCP segments; overlapping duplicate segments with garbage payloads (server-side exploit)	PASS
small TCP segments in reverse order (server-side exploit)	PASS
small TCP segments in random order (server-side exploit)	PASS
small TCP segments; delay first segment (server-side exploit)	PASS
small TCP segments in reverse order; delay last segment (server-side exploit)	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay first segment (server-side exploit)	PASS
small TCP segments in random order; interleave chaff before (invalid TCP checksums); delay random segment (server-side exploit)	PASS
small TCP segments in random order; interleave chaff sandwich (out-of-window sequence numbers); TCP MSS option (server-side exploit)	PASS
small TCP segments in random order; interleave chaff after (requests to resynch sequence numbers mid-stream); TCP window scale option (server-side exploit)	PASS
small TCP segments in random order; interleave chaff sandwich (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment (server-side exploit)	PASS
small overlapping TCP segments (server-side exploit)	PASS
small overlapping TCP segments; method 2 (server-side exploit)	PASS
small overlapping TCP segments; method 3 (server-side exploit)	PASS
small TCP segments; small IP fragments (server-side exploit)	PASS
small TCP segments; small IP fragments in reverse order (server-side exploit)	PASS
small TCP segments in random order; small IP fragments (server-side exploit)	PASS
small TCP segments; small IP fragments in random order (server-side exploit)	PASS
small TCP segments in random order; small IP fragments in reverse order (server-side exploit)	PASS
small TCP segments in random order; interleave chaff sandwich (invalid TCP checksums); small overlapping IP fragments in reverse order; interleave chaff after (invalid IP options) (server-side exploit)	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IP fragments; interleave chaff before (invalid IP options) (server-side exploit)	PASS
small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IP fragments; interleave chaff sandwich (invalid IP options); delay last fragment (server-side exploit)	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 (server-side exploit)	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 (server-side exploit)	PASS
small overlapping TCP segments; overlapping small fragments (server-side exploit)	PASS
small overlapping TCP segments; delay last segment; overlapping small fragments; delay last fragment (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid length) (server-side exploit)	PASS
small TCP segments; interleave chaff sandwich (invalid IP options; invalid loose source route pointer points past empty address field) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points before first address) (server-side exploit)	PASS

TCP Segmentation	Results
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points past last address) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points to middle of first address) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; more than two loose source route options) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points before first address) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points past last address)) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points to middle of first address) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; more than two strict source route options) (server-side exploit)	PASS
small TCP segments; interleave chaff sandwich (invalid IP options; invalid strict source route pointer points past empty address field) (server-side exploit)	PASS
small TCP segments; over IPv6 (server-side exploit)	PASS
small TCP segments in reverse order; over IPv6 (server-side exploit)	PASS
small TCP segments in random order; over IPv6 (server-side exploit)	PASS
small TCP segments; delay first segment; over IPv6 (server-side exploit)	PASS
small TCP segments in reverse order; delay last segment; over IPv6 (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid TCP checksums); delay first segment; over IPv6 (server-side exploit)	PASS
small TCP segments in random order; interleave chaff after (older PAWS timestamps); delay last segment; over IPv6 (server-side exploit)	PASS
small TCP segments in random order; interleave chaff before (out-of-window sequence numbers); TCP MSS option; over IPv6 (server-side exploit)	PASS
small TCP segments in random order; interleave chaff sandwich (requests to resynch sequence numbers mid-stream); TCP window scale option; over IPv6 (server-side exploit)	PASS
small TCP segments in random order; interleave chaff before (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment; over IPv6 (server-side exploit)	PASS
small overlapping TCP segments; over IPv6 (server-side exploit)	PASS
small TCP segments; small IPv6 fragments (server-side exploit)	PASS
small TCP segments; small IPv6 fragments in reverse order (server-side exploit)	PASS
small TCP segments in random order; small IPv6 fragments (server-side exploit)	PASS
small TCP segments; small IPv6 fragments in random order (server-side exploit)	PASS
small TCP segments in random order; small IPv6 fragments in reverse order (server-side exploit)	PASS
small TCP segments in random order; interleave chaff before (invalid TCP checksums); small IPv6 fragments in reverse order (server-side exploit)	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IPv6 fragments (server-side exploit)	PASS
small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IPv6 fragments; delay last fragment (server-side exploit)	PASS
small TCP segments in random order; interleave chaff sandwich (out-of-window sequence numbers); small IPv6 fragments in random order; delay random fragment (server-side exploit)	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 (server-side exploit)	PASS
small overlapping TCP segments; small IPv6 fragments (server-side exploit)	PASS
small overlapping TCP segments; delay last segment; small IPv6 fragments; delay last fragment (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; IPv6 Invalid Destination Options Extension Header) (server-side exploit)	PASS

TCP Segmentation	Results
open TCP session and wait 61 minutes to send exploit (server-side exploit)	PASS
open TCP session and send small pieces of application protocol headers; pausing between each piece (server-side exploit)	PASS
open TCP session and send small pieces of application protocol headers; pausing between each piece; over IPv6 (server-side exploit)	PASS

Figure 6 – TCP Segmentation Results

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 on 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.

An attacker may be able to evade detection by **segmenting the TCP streams** in any number of ways, such as sending them in reverse 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.

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 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.

RPC Fragmentation

Both Sun/ONC RPC and MS-RPC allow the sending application to fragment requests, and all MS-RPC services have a built-in fragmentation reassembly mechanism.

An attacker can transmit the BIND followed by a single request fragmented over a hundred actual requests with small fragments of the malicious payload. Alternatively, the attacker could transmit both the BIND and request fragments in one large TCP segment, thus foiling any signatures that use a simple size check.

RPC Fragmentation	Results
One-byte fragmentation (ONC)	PASS
Two-byte fragmentation (ONC)	PASS
All fragments, including Last Fragment (LF) will be sent in one TCP segment (ONC)	PASS
All frags except Last Fragment (LF) will be sent in one TCP segment. LF will be sent in separate TCP seg (ONC)	PASS
One RPC fragment will be sent per TCP segment (ONC)	PASS
One LF split over more than one TCP segment. In this case no RPC fragmentation is performed (ONC)	PASS
Canvas Reference Implementation Level 1 (MS)	PASS
Canvas Reference Implementation Level 2 (MS)	PASS
Canvas Reference Implementation Level 3 (MS)	PASS
Canvas Reference Implementation Level 4 (MS)	PASS
Canvas Reference Implementation Level 5 (MS)	PASS
Canvas Reference Implementation Level 6 (MS)	PASS
Canvas Reference Implementation Level 7 (MS)	PASS
Canvas Reference Implementation Level 8 (MS)	PASS

RPC Fragmentation	Results
Canvas Reference Implementation Level 9 (MS)	PASS
Canvas Reference Implementation Level 10 (MS)	PASS

Figure 7 – RPC Fragmentation Results

URL Obfuscation

Random URL encoding techniques are employed to transform simple URLs that are often used in pattern-matching signatures into apparently meaningless strings of escape sequences and expanded path characters using a combination of the following techniques:

- Escape encoding (% encoding)
- Microsoft %u encoding
- Path character transformations and expansions (/./ , //, \ )

These techniques are combined in various ways for each URL tested, ranging from minimal transformation to extreme (every character transformed). All transformed URLs are verified to ensure they still function as expected after transformation.

URL Obfuscation	Results
URL encoding – Level 1 (minimal)	PASS
URL encoding – Level 2	PASS
URL encoding – Level 3	PASS
URL encoding – Level 4	PASS
URL encoding – Level 5	PASS
URL encoding – Level 6	PASS
URL encoding – Level 7	PASS
URL encoding – Level 8 (extreme)	PASS
Directory Insertion	PASS
Premature URL ending	PASS
Long URL	PASS
Fake parameter	PASS
TAB separation	PASS
Case sensitivity	PASS
Windows \ delimiter	PASS
Session splicing	PASS

Figure 8 – URL Obfuscation Results

FTP & Telnet Evasions

When attempting FTP and Telnet exploits, it is possible to evade some deep inspection products by inserting additional spaces and Telnet control sequences in FTP and Telnet commands.

These tests insert a range of valid Telnet control sequences that can be parsed and that handle legitimate services that conform to RFCs. Control opcodes are then inserted at random, ranging from minimal insertion (only one pair of opcodes), to extreme (opcodes between every character in the command).

FTP & Telnet Evasion	Results
Inserting spaces in FTP command lines	PASS
Inserting non-text Telnet opcodes – Level 1 (minimal)	PASS
Inserting non-text Telnet opcodes – Level 2	PASS
Inserting non-text Telnet opcodes – Level 3	PASS
Inserting non-text Telnet opcodes – Level 4	PASS
Inserting non-text Telnet opcodes – Level 5	PASS
Inserting non-text Telnet opcodes – Level 6	PASS
Inserting non-text Telnet opcodes – Level 7	PASS
Inserting non-text Telnet opcodes – Level 8 (extreme)	PASS

Figure 9 – Telnet and FTP Evasions Results

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.

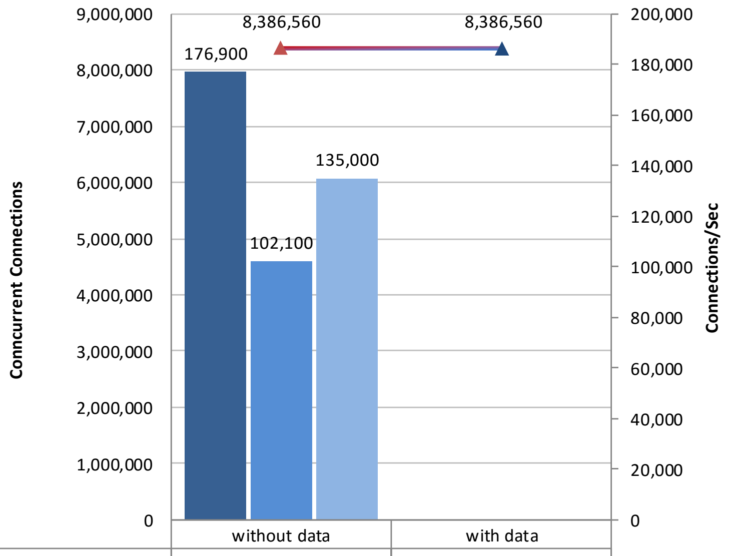
When considering a security device for the data center, knowing the theoretical limits and connection dynamics is key. A data center security device needs to support much higher concurrent connections, connections per second, and transactions per second as it handles traffic for a large number of users who are accessing applications in a private cloud. Stateless UDP traffic (such as that seen in a network file system [NFS]) and long-lived transmission control protocol (TCP) connections {as would be seen in an iSCSI storage area network [SAN] or backup application) present a continuous and heavy load. Finally, excessive latency can prevent sensitive applications from functioning properly.

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



	without data	with data
TCP Connections per Second	176,900	
HTTP Connections per Second	102,100	
HTTP Transactions per Second	135,000	
Concurrent TCP Connections	8,386,560	8,386,560

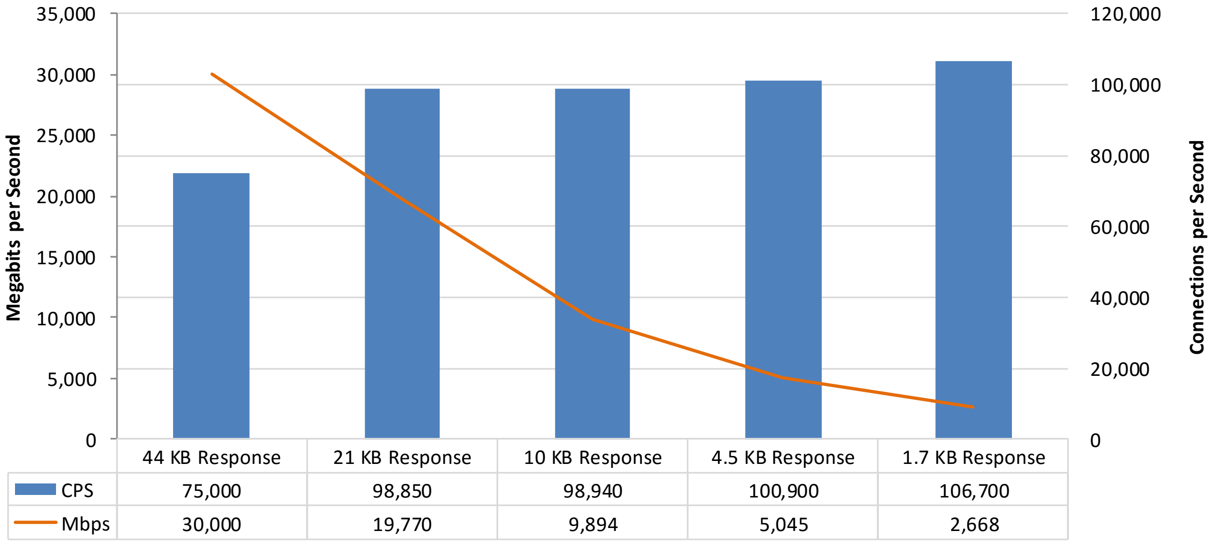
Figure 10 – Maximum Capacity (Concurrency and Connection Rates)

HTTP Capacity

The aim of this test 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. 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.

Figure 11 depicts the results of the HTTP capacity with no transaction delays test.



	44 KB Response	21 KB Response	10 KB Response	4.5 KB Response	1.7 KB Response
CPS	75,000	98,850	98,940	100,900	106,700
Mbps	30,000	19,770	9,894	5,045	2,668

Figure 11 – HTTP Capacity with No Transaction Delays

Application Average Response Time – HTTP

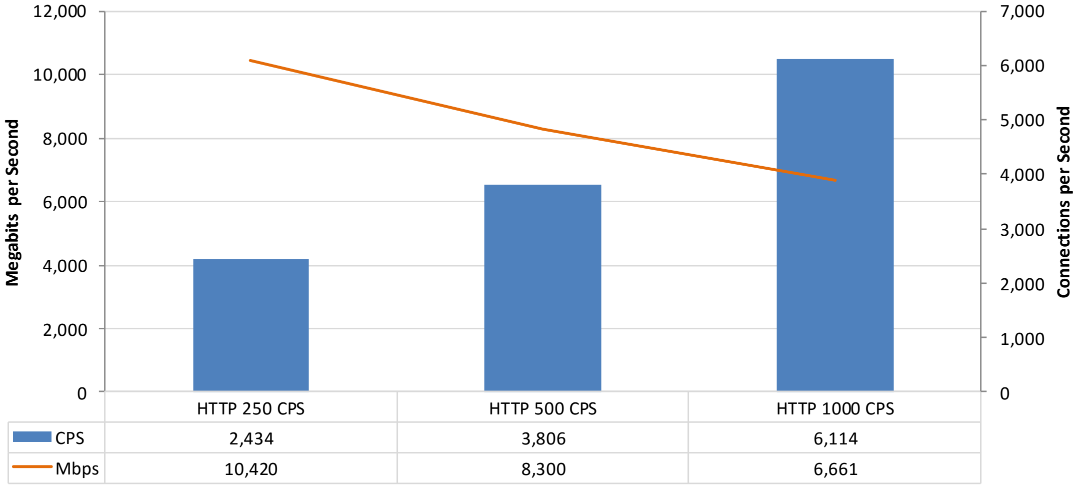
Application Average Response Time – HTTP (at 95% Maximum Load)	Results
2,500 Connections per Second – 44-KB Response	1.51
5,000 Connections per Second – 21-KB Response	1.21
10,000 Connections per Second – 10-KB Response	0.98
20,000 Connections per Second – 4.5-KB Response	0.87
40,000 Connections per Second – 1.7-KB Response	1.13

Figure 12 – Average Application Response Time (Milliseconds)

HTTP Capacity with HTTP Persistent Connections

The aim of this test is to determine how the DCSG copes with network loads of varying average packet size and varying connections per second while inspecting traffic. By creating genuine session-based traffic with varying session lengths, the DCSG 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 it is possible to achieve in a lab environment, while ensuring absolute accuracy and repeatability.

This test will use HTTP persistent connections, with each TCP connection containing 10 HTTP GETs and associated responses. All packets contain valid payload (a mix of binary and ASCII objects) and address data, and this test provides an excellent representation of a live network at various network loads. The stated response size is the total of all HTTP responses within a single TCP session. Figure 13 depicts the results of the HTTP capacity with HTTP persistent connections test.

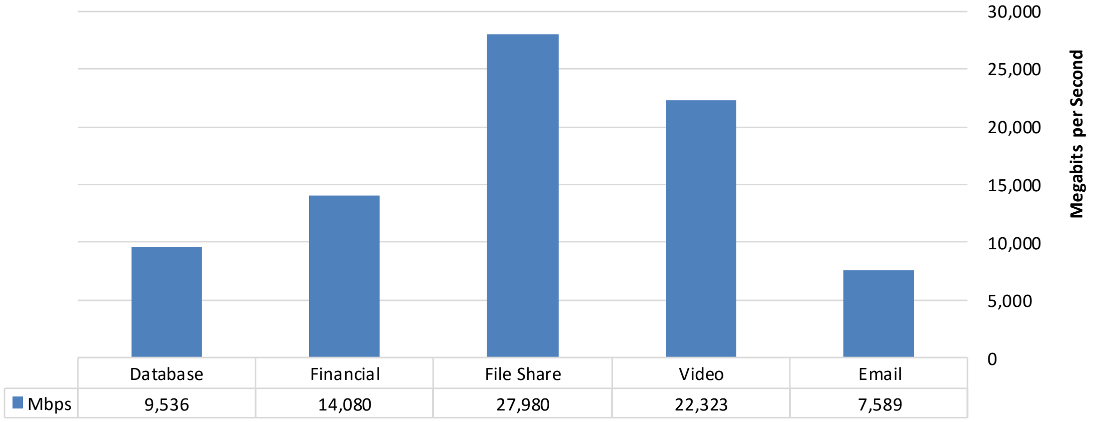


	HTTP 250 CPS	HTTP 500 CPS	HTTP 1000 CPS
CPS	2,434	3,806	6,114
Mbps	10,420	8,300	6,661

Figure 13 – HTTP Capacity with HTTP Persistent Connections

Single Application Flows

This test measures the performance of the device with single application flows. For details about single application flow testing, see the NSS Labs Data Center Network Security (DCNS) Test Methodology v3.1, available at www.nsslabs.com. Figure 14 depicts the results of the single application flows test.



	Database	Financial	File Share	Video	Email
Mbps	9,536	14,080	27,980	22,323	7,589

Figure 14 – Single Application Flows

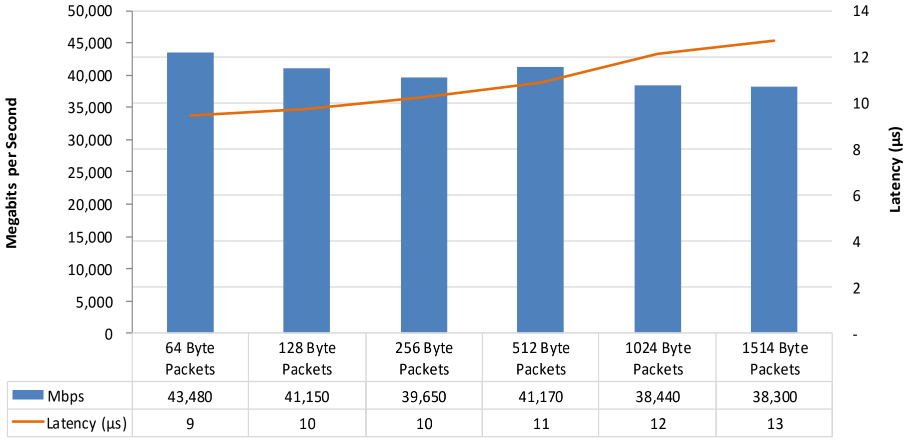
Raw Packet Processing Performance (UDP Throughput)

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 any real-world network condition. No TCP sessions are created during this test, and there is very little for the detection engine to do. However, each vendor is required to write a signature to detect the test packets in order to ensure that they are being passed through the detection engine and are not being “fast-pathed.”

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 lowest amount of latency.



	64 Byte Packets	128 Byte Packets	256 Byte Packets	512 Byte Packets	1024 Byte Packets	1514 Byte Packets
Mbps	43,480	41,150	39,650	41,170	38,440	38,300
Latency (µs)	9	10	10	11	12	13

Figure 15 – Raw Packet Processing Performance – UDP Traffic

Raw Packet Processing Performance (UDP Latency)

DCSGs 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 16 depicts UDP latency (in microseconds) as recorded during the UDP throughput tests at 95% of the maximum load. UDP was tested over IPv4.

Latency – UDP	Results
64-Byte Packets	9.42
128-Byte Packets	9.74
256-Byte Packets	10.22
512-Byte Packets	10.89
1024-Byte Packets	12.11
1514-Byte Packets	12.71

Figure 16 – UDP Latency in Microseconds

NSS-Tested Throughput: Use Cases

Because data center network traffic can vary greatly between industries and enterprises, NSS has created three separate use cases. Each use case weights test results in order to align them with the different use cases seen in a data center, i.e., transactional, multimedia, or corporate.

The corporate use case may be best described as the data center footprint of a typical enterprise, where mission-critical applications such as email and ERP (enterprise resource planning software) are kept. **The rated throughput emphasizes various packet sizes and protocols that are more likely to be found in those situations, such as email, database, and file sharing.**

The transactional use case reflects a data center with traffic that is more transactional in nature. An example of this may include B2B (business-to-business) or B2C (business-to-consumer) e-commerce. **The rated throughput emphasizes smaller packet sizes and connections per second.**

The multimedia use case reflects a data center whose purpose is to serve media content. **The rated throughput emphasizes larger packet sizes, maximum concurrent sessions, and streaming protocols.**

Use Case	Results
Transactional (small packets, database, email)	9,467 Mbps
Multimedia (video, large packets, database, email)	16,135 Mbps
Corporate (email, file share, database, mix of packet sizes)	12,842 Mbps

Figure 17 – NSS-Tested Throughput: Use Cases

Stability & Reliability

Long-term stability is particularly important for an inline device, since 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 inspecting malicious and non-malicious traffic. Products that cannot sustain legitimate traffic (or that crash) while under hostile attack will not pass. Stability & reliability was tested over IPv4 and IPv6.

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, the device will fail the test.

Stability & Reliability	Result
Attack Detection/Blocking – Normal Load	PASS
State Preservation – Normal Load	PASS
Pass Legitimate Traffic – Normal Load	PASS
State Preservation – Maximum Exceeded	PASS
Power Fail	PASS
Persistence of Data	PASS
High Availability (HA) – Optional	FAIL

Figure 18 – Stability & Reliability Results

These tests also determine the behavior of the state engine under load. A DCSG device 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 DCSG will block legitimate traffic but maintain state on existing connections (and prevent attack leakage).

Total Cost of Ownership (TCO)

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

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

Installation Time

Installation Time is the number of hours of labor required to install each device, using only local management options. This is the amount of time taken by NSS engineers and vendor engineers to install and configure the device to the point where it operated successfully in the test harness, passed legitimate traffic, and prohibited/malicious traffic.

Product	Installation (Hours)
Palo Alto Networks PA-5250 9.0.3-h2	8

Figure 19 – Device Installation Time (Hours)

Total Cost of Ownership

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

Product	Purchase Price	Maintenance/ Year	Year 1 Cost	Year 2 Cost	Year 3 Cost	3-Year TCO
Palo Alto Networks PA-5250 9.0.3-h2	$75,000	$21,600	$97,200	$21,600	$21,600	$140,400

Figure 20 – 3-Year TCO (US$)

- **Year 1 Cost** is calculated by adding installation costs (US$75 per hour fully loaded labor x installation time) + purchase price + first-year maintenance/support fees.
- **Year 2 Cost** consists only of maintenance/support fees.
- **Year 3 Cost** consists only of maintenance/support fees.

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

Appendix A: Product Scorecard

Security Effectiveness	
Block Rate	99.45%
False Positive Testing	PASS
Evasions and Attack Leakage	
IP Packet Fragmentation	
small IP fragments; overlapping duplicate fragments with garbage payloads (server-side exploit)	PASS
small overlapping IP fragments in reverse order (server-side exploit)	PASS
small overlapping IP fragments in random order (server-side exploit)	PASS
small IP fragments; delay first fragment (server-side exploit)	PASS
small IP fragments in reverse order; delay last fragment (server-side exploit)	PASS
small IP fragments; interleave chaff after (invalid IP options) (server-side exploit)	PASS
small IP fragments in random order; interleave chaff sandwich (invalid IP options) (server-side exploit)	PASS
small overlapping IP fragments in random order; interleave chaff sandwich (invalid IP options); delay random fragment (server-side exploit)	PASS
small IP fragments; interleave chaff before (invalid IP options); DSCP value 16 (server-side exploit)	PASS
small IP fragments in random order; interleave chaff after (invalid IP options); delay random fragment; DSCP value 34 (server-side exploit)	PASS
IPv4 fragmentation with an overlapping atomic fragment with good data inserted in-between the fragments with junk data (server-side exploit)	PASS
IPv4 fragmentation with an overlapping atomic fragment with junk data inserted in-between the fragments with good data (server-side exploit)	PASS
small IPv6 fragments (server-side exploit)	PASS
small IPv6 fragments in reverse order (server-side exploit)	PASS
small IPv6 fragments in random order (server-side exploit)	PASS
small IPv6 fragments; delay first fragment (server-side exploit)	PASS
small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay first fragment (server-side exploit)	PASS
small IPv6 fragments in reverse order; delay last fragment (server-side exploit)	PASS
small IPv6 fragments in reverse order; interleave duplicate fragments with garbage payloads; delay random fragment (server-side exploit)	PASS
small IPv6 fragments in random order; delay first fragment (server-side exploit)	PASS
small IPv6 fragments in random order; delay last fragment (server-side exploit)	PASS
small IPv6 fragments in random order; delay random fragment (server-side exploit)	PASS
TCP Segmentation	
small TCP segments; overlapping duplicate segments with garbage payloads (server-side exploit)	PASS
small TCP segments in reverse order (server-side exploit)	PASS
small TCP segments in random order (server-side exploit)	PASS
small TCP segments; delay first segment (server-side exploit)	PASS
small TCP segments in reverse order; delay last segment (server-side exploit)	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay first segment (server-side exploit)	PASS
small TCP segments in random order; interleave chaff before (invalid TCP checksums); delay random segment (server-side exploit)	PASS
small TCP segments in random order; interleave chaff sandwich (out-of-window sequence numbers); TCP MSS option (server-side exploit)	PASS
small TCP segments in random order; interleave chaff after (requests to resynch sequence numbers mid-stream); TCP window scale option (server-side exploit)	PASS

small TCP segments in random order; interleave chaff sandwich (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment (server-side exploit)	PASS
small overlapping TCP segments (server-side exploit)	PASS
small overlapping TCP segments; method 2 (server-side exploit)	PASS
small overlapping TCP segments; method 3 (server-side exploit)	PASS
small TCP segments; small IP fragments (server-side exploit)	PASS
small TCP segments; small IP fragments in reverse order (server-side exploit)	PASS
small TCP segments in random order; small IP fragments (server-side exploit)	PASS
small TCP segments; small IP fragments in random order (server-side exploit)	PASS
small TCP segments in random order; small IP fragments in reverse order (server-side exploit)	PASS
small TCP segments in random order; interleave chaff sandwich (invalid TCP checksums); small overlapping IP fragments in reverse order; interleave chaff after (invalid IP options) (server-side exploit)	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IP fragments; interleave chaff before (invalid IP options) (server-side exploit)	PASS
small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IP fragments; interleave chaff sandwich (invalid IP options); delay last fragment (server-side exploit)	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 (server-side exploit)	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 (server-side exploit)	PASS
small overlapping TCP segments; overlapping small fragments (server-side exploit)	PASS
small overlapping TCP segments; delay last segment; overlapping small fragments; delay last fragment (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid length) (server-side exploit)	PASS
small TCP segments; interleave chaff sandwich (invalid IP options; invalid loose source route pointer points past empty address field) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points before first address) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points past last address) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid loose source route pointer points to middle of first address) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; more than two loose source route options) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points before first address) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points past last address)) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; invalid strict source route pointer points to middle of first address) (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; more than two strict source route options) (server-side exploit)	PASS
small TCP segments; interleave chaff sandwich (invalid IP options; invalid strict source route pointer points past empty address field) (server-side exploit)	PASS
small TCP segments; over IPv6 (server-side exploit)	PASS
small TCP segments in reverse order; over IPv6 (server-side exploit)	PASS
small TCP segments in random order; over IPv6 (server-side exploit)	PASS
small TCP segments; delay first segment; over IPv6 (server-side exploit)	PASS
small TCP segments in reverse order; delay last segment; over IPv6 (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid TCP checksums); delay first segment; over IPv6 (server-side exploit)	PASS
small TCP segments in random order; interleave chaff after (older PAWS timestamps); delay last segment; over IPv6 (server-side exploit)	PASS

small TCP segments in random order; interleave chaff before (out-of-window sequence numbers); TCP MSS option; over IPv6 (server-side exploit)	PASS
small TCP segments in random order; interleave chaff sandwich (requests to resynch sequence numbers mid-stream); TCP window scale option; over IPv6 (server-side exploit)	PASS
small TCP segments in random order; interleave chaff before (requests to resynch sequence numbers mid-stream); TCP window scale option; delay first segment; over IPv6 (server-side exploit)	PASS
small overlapping TCP segments; over IPv6 (server-side exploit)	PASS
small TCP segments; small IPv6 fragments (server-side exploit)	PASS
small TCP segments; small IPv6 fragments in reverse order (server-side exploit)	PASS
small TCP segments in random order; small IPv6 fragments (server-side exploit)	PASS
small TCP segments; small IPv6 fragments in random order (server-side exploit)	PASS
small TCP segments in random order; small IPv6 fragments in reverse order (server-side exploit)	PASS
small TCP segments in random order; interleave chaff before (invalid TCP checksums); small IPv6 fragments in reverse order (server-side exploit)	PASS
small TCP segments; interleave chaff after (invalid TCP checksums); delay last segment; small IPv6 fragments (server-side exploit)	PASS
small TCP segments; interleave chaff sandwich (invalid TCP checksums); small IPv6 fragments; delay last fragment (server-side exploit)	PASS
small TCP segments in random order; interleave chaff sandwich (out-of-window sequence numbers); small IPv6 fragments in random order; delay random fragment (server-side exploit)	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 (server-side exploit)	PASS
small overlapping TCP segments; small IPv6 fragments (server-side exploit)	PASS
small overlapping TCP segments; delay last segment; small IPv6 fragments; delay last fragment (server-side exploit)	PASS
small TCP segments; interleave chaff (invalid IP options; IPv6 Invalid Destination Options Extension Header) (server-side exploit)	PASS
open TCP session and wait 61 minutes to send exploit (server-side exploit)	PASS
open TCP session and send small pieces of application protocol headers; pausing between each piece (server-side exploit)	PASS
open TCP session and send small pieces of application protocol headers; pausing between each piece; over IPv6 (server-side exploit)	PASS
Resiliency	
• Payload	
Python one-liner bind shell listener for post exploitation payload; rather than nc in base exploit (server-side exploit)	PASS
• Trigger	
Payload is executed using the system () PHP function (server-side exploit)	PASS
Payload is executed using the passthru () PHP function (server-side exploit)	PASS
• White Space	
Whitespace added in the "mail[#markup]" field before the payload (server-side exploit)	PASS
• Payload + Whitespace	
Combination of methods used in sres-wsp-001; and sres-pay-001 (server-side exploit)	PASS
• Payload + Trigger + Whitespace	
Combination of methods used in sres-wsp-001; sres-pay-001; and sres-trg-001 (server-side exploit)	PASS
RPC Fragmentation	
One-byte fragmentation (ONC)	PASS
Two-byte fragmentation (ONC)	PASS
All fragments, including Last Fragment (LF) will be sent in one TCP segment (ONC)	PASS
All frags except Last Fragment (LF) will be sent in one TCP segment. LF will be sent in separate TCP seg (ONC)	PASS
One RPC fragment will be sent per TCP segment (ONC)	PASS

One LF split over more than one TCP segment. In this case no RPC fragmentation is performed (ONC)	PASS
Canvas Reference Implementation Level 1 (MS)	PASS
Canvas Reference Implementation Level 2 (MS)	PASS
Canvas Reference Implementation Level 3 (MS)	PASS
Canvas Reference Implementation Level 4 (MS)	PASS
Canvas Reference Implementation Level 5 (MS)	PASS
Canvas Reference Implementation Level 6 (MS)	PASS
Canvas Reference Implementation Level 7 (MS)	PASS
Canvas Reference Implementation Level 8 (MS)	PASS
Canvas Reference Implementation Level 9 (MS)	PASS
Canvas Reference Implementation Level 10 (MS)	PASS
URL Obfuscation	
URL encoding – Level 1 (minimal)	PASS
URL encoding – Level 2	PASS
URL encoding – Level 3	PASS
URL encoding – Level 4	PASS
URL encoding – Level 5	PASS
URL encoding – Level 6	PASS
URL encoding – Level 7	PASS
URL encoding – Level 8 (extreme)	PASS
Directory Insertion	PASS
Premature URL ending	PASS
Long URL	PASS
Fake parameter	PASS
TAB separation	PASS
Case sensitivity	PASS
Windows \ delimiter	PASS
Session splicing	PASS
FTP Evasion	
Inserting spaces in FTP command lines	PASS
Telnet Evasions	
Inserting non-text Telnet opcodes – Level 1 (minimal)	PASS
Inserting non-text Telnet opcodes – Level 2	PASS
Inserting non-text Telnet opcodes – Level 3	PASS
Inserting non-text Telnet opcodes – Level 4	PASS
Inserting non-text Telnet opcodes – Level 5	PASS
Inserting non-text Telnet opcodes – Level 6	PASS
Inserting non-text Telnet opcodes – Level 7	PASS
Inserting non-text Telnet opcodes – Level 8 (extreme)	PASS
Performance	
Raw Packet Processing Performance (UDP Traffic) (IPV4 Only)	Mbps
64-Byte Packets	43,480
128-Byte Packets	41,150
256-Byte Packets	39,650

512-Byte Packets	41,170
1,024-Byte Packets	38,440
1,514-Byte Packets	38,300
Latency – UDP	Microseconds
64-Byte Packets	9.42
128-Byte Packets	9.74
256-Byte Packets	10.22
512-Byte Packets	10.89
1,024-Byte Packets	12.11
1,514-Byte Packets	12.71
Maximum Capacity	
Theoretical Maximum Concurrent TCP Connections	8,386,560
Maximum TCP Connections per Second	176,900
Maximum HTTP Connections per Second	102,100
Maximum HTTP Transactions per Second	135,000
HTTP Capacity with No Transaction Delays	
25,000 Connections per Second – 44-KB Response	75,000
50,000 Connections per Second – 21-KB Response	98,850
100,000 Connections per Second – 10-KB Response	98,940
200,000 Connections per Second – 4.5-KB Response	100,900
400,000 Connections per Second – 1.7-KB Response	106,700
Application Average Response Time Maximum HTTP (at 95% Max Load)	Milliseconds
25,000 Connections per Second – 44-KB Response	1.51
50,000 Connections per Second – 21-KB Response	1.21
100,000 Connections per Second – 10-KB Response	0.98
200,000 Connections per Second – 4.5-KB Response	0.87
400,000 Connections per Second – 1.7-KB Response	1.13
HTTP Capacity with HTTP Persistent Connections	CPS
250 Connections per Second	2,434
500 Connections per Second	3,806
1,000 Connections per Second	6,114
Single Application Flows	Mbps
Database	9,536
Financial	14,080
File Share	27,980
Video	22,323
Email	7,589
Stability & Reliability	
Blocking Under Extended Load with Attacks	PASS
Behavior of the State Engine under Load	PASS
State Preservation – Normal Load	PASS
State Preservation – Maximum Exceeded	PASS
Power Fail	PASS
Persistence of Data	PASS

High Availability (HA) – Optional	FAIL
Total Cost of Ownership	
Ease of Use	
Initial Setup (Hours)	8
Time Required for Upkeep (Hours per Year)	Contact NSS Labs
Time Required to Tune (Hours per Year)	Contact NSS Labs
Expected Costs	
Initial Purchase (hardware as tested)	$75,000
Installation Labor Cost (@$75/hr)	$600
Annual Cost of Maintenance & Support (hardware/software)	$21,600
Annual Cost of Updates (IPS/AV/etc.)	$0
Initial Purchase (centralized management system)	Contact NSS Labs
Annual Cost of Maintenance & Support (centralized management system)	Contact NSS Labs
Management Labor Cost (per Year @$75/hr)	Contact NSS Labs
Tuning Labor Cost (per Year @$75/hr)	Contact NSS Labs
Total Cost of Ownership	
Year 1	$97,200
Year 2	$21,600
Year 3	$21,600
3-Year Total Cost of Ownership	$141,400

Figure 21 – Detailed Scorecard

Test Methodology

Data Center Network Security (DCNS) Test Methodology v3.1

Evasions Test Methodology v1.1

Copies of the test methodologies are 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

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

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

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

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

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

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

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

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