



TOMAHAWK

**A Methodology and Toolset
for Evaluating Network
Based Intrusion Prevention
Systems**

Abstract

Although network-based Intrusion Prevention Systems (NIPS) have recently entered the commercial marketplace as an important defense component security strategy, methodologies for evaluating these systems are still quite primitive. Few commercial tools are available for evaluating the security functionality of NIPS, and TippingPoint believes that current commercial tools for testing the network performance of these devices do not effectively test and report a network's protection. As part of its internal testing and quality assurance programs, TippingPoint has developed a suite of tools for testing our NIPS over the past three years. This paper, and the accompanying software, present the results of this effort and provide powerful tools for evaluating NIPS.

Introduction

Network-based Intrusion Prevention Systems (NIPS) are inline network devices that detect and block network-based attacks. A NIPS is deployed in a network much like a switch or a router. The NIPS inspects every packet that passes through it, scanning for attacks designed to infect, disable, or take over another computer system. When the NIPS detects an attack, it takes an action, typically blocking the corresponding packet stream and generating a notification.

As an element of the network infrastructure, a NIPS must perform like switches or routers in latency and throughput. It must also identify attacks without blocking legitimate traffic and detect attacks despite the use of evasion techniques [1]. As the guardian of the network, the NIPS itself must be extremely resilient against attacks lest it become a target for Denial of Service attacks.

Although NIPS are an important component of a defensive security strategy, methodologies for evaluating these systems are still quite primitive. Customers and other evaluators typically rely on test suites based on some combination of commercial, open source, and "home grown" tools. These suites are often aggregated from what is easily available, without regard to their effectiveness.

Over the past three years, TippingPoint has developed a powerful set of tools for evaluating the network and security performance of NIPS. This paper describes these tools and their use in evaluating NIPS products. Our methodology for testing NIPS involves two types of tests: Network Performance and Security Performance.

Network and Application Performance Testing

Network Performance testing checks the performance of the NIPS as an element of the network infrastructure by measuring network latency and throughput under a variety of traffic loads. Ideally, the latency and throughput of the NIPS should be on par with other network elements in which it is installed. For example, if the NIPS is installed in a LAN, the one-way latency should be less than 200 microseconds and the throughput 1 Gbps or more. If the NIPS is installed in a campus area network, the one-way latency can rise to 500 microseconds. At the perimeter (Internet connection), higher latencies and lower throughput may be acceptable.

Application Performance testing investigates the effect of the NIPS on the performance of network intensive applications, such as NFS file copy and web file retrieval. Performance is measured by calculating the time to complete an application workload (e.g., copying a large file) with and without the NIPS in place. The effect of the NIPS can then be measured as a percentage slowdown. Ideally, the effect of the NIPS on application performance should be negligible.

Security Performance

Security Performance testing evaluates the NIPS security functions. This includes the NIPS ability to detect and block attacks, false positive resistance, and evasion resistance. An important component of this testing is *repeatability* testing: given the same attack repeated many times, the test checks that the NIPS consistently detects and blocks the attack.

The Tomahawk Test Tool and Jig

The Tomahawk Test Tool

Our test jig leverages the *Tomahawk* test tool, software developed by TippingPoint for testing both the network and security performance of NIPSSs.

The following diagram shows a single Tomahawk server connected to a NIPS. The Tomahawk server is run on a Linux machine with three network interface cards (NICs): one for management and two for testing. The two test NICs are eth0 and eth1; the management NIC is eth2. The two test NICs are connected, typically through a switch, crossover cable, or IPS. The network connecting the two test NICs must be a layer-2 network.

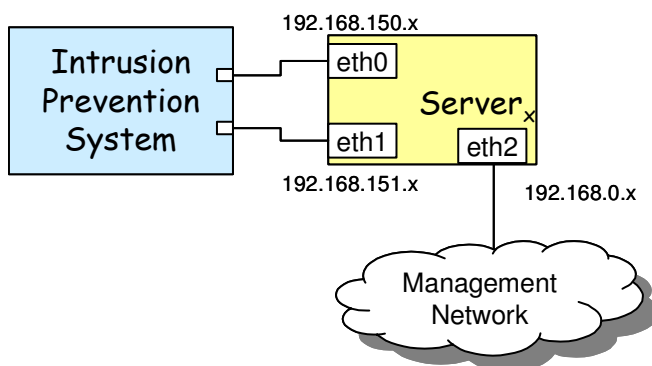


Figure 1: Configuration of a single Tomahawk traffic server

Tomahawk is used to replay one or more *PCAP files* through a NIPS. PCAP files are packet traces captured in tcpdump format by a network sniffer. When Tomahawk replays a PCAP file, the packets in the PCAP arrive on the NIPS interfaces in the same order as the NIPS would have seen had it been installed inline when the PCAP was captured.

For example, when an attacker creates a TCP connection to a victim, the two complete a three-way handshake consisting of three packets: SYN, SYN-ACK and ACK [2]. Suppose this interaction had been captured in a PCAP file. When this PCAP is replayed using Tomahawk, Tomahawk will first transmit the SYN packet on eth0 and wait for the packet to arrive on eth1.

When this happens, Tomahawk transmits the SYN-ACK packet on eth1. When the SYN-ACK packet arrives on eth0, Tomahawk transmits the SYN-ACK packet on eth0.

If any packet is lost in transit, Tomahawk retransmits it after a timeout. If the packet does not arrive after a user-specified number of retransmissions, Tomahawk reports that the PCAP has “timed out” on the management console. If no packets timeout, Tomahawk reports that the PCAP has completed.

The throughput of the system just described is limited by the latency of the NIPS. For example, if the latency is one second, then the three-way handshake will complete at one packet per second. To overcome this limitation, Tomahawk uses two techniques: pipelining and parallel replay.

Pipelining improves performance by transmitting as many packets at once as possible. That is, Tomahawk will send 10 packets in a burst. In the example above, this would increase the throughput to ten packets per second. The degree of pipelining is limited by the constraint that the traffic arrival order at the NIPS interfaces must be consistent with the packet order in the PCAP. For example, it would be inconsistent to send the SYN and SYN-ACK in a burst, because in a real client the receipt of the SYN triggers the transmission of the SYN-ACK. Sending them in a burst could result in the SYN-ACK being received at the NIPS interface before the SYN had been received, a situation that could not happen in real networks.

Parallel replay further improves performance by playing multiple copies of each traffic trace. Each copy is given its own unique range of IP addresses. For example, two copies a three-way handshake can be played in parallel by given the first copy one pair of addresses and the second copy a different pair. To the NIPS, the traffic appears to be generated by four machines: two servers and two clients. Tomahawk can play thousands of copies of streams in parallel¹.

Tomahawk can also be used to generate load by replaying clean, background traffic. Traffic can be captured from a customer network and replayed using Tomahawk to evaluate how well the NIPS will perform with that traffic mix. The combination of pipelining and parallel replay allows Tomahawk to simulate a high-speed network with tens of thousands of individual hosts. This traffic will have the same protocol and payload mix as the traffic trace that is replayed. A typical Tomahawk traffic server can generate up to 400 Mbps of load, so by using a few traffic servers and switches, network loads of several gigabits can be created.

When Tomahawk replays a PCAP containing an attack, it can be used to test NIPS security functions as a “black box”, i.e., without parsing the NIPS alert logs. Any PCAP that contains an attack that the NIPS purportedly blocks should time out. If it does not time out, then the NIPS failed to block the attack (a false negative), regardless of messages any log file indicates. Conversely, if a PCAP does not contain an attack and the NIPS blocks it, then the NIPS has blocked legitimate traffic (a false positive).

¹ Appendix A contains further details on the engineering and use of Tomahawk.

Because it can replay the same attack thousands of times, each with a unique IP address, Tomahawk can be used for repeatability testing. This test ensures that an attack is consistently blocked by the NIPS.

The Tomahawk Jig

By aggregating the traffic from multiple Tomahawk traffic servers, several gigabits/sec of network traffic can be generated and other tests, such as network latency and application performance, can be executed. The jig to perform this testing is called the *Tomahawk jig*, and is shown in figure 2. For clarity, the management network (eth2) is not shown.

On each traffic server, eth0 interfaces are assigned IP addresses out of the 192.168.150.0/24 subnet; eth1 interfaces are assigned IP addresses out of the 192.168.151.0/24 subnet. The management interfaces (eth2) are assigned IP addresses out of the 192.168.0.0/24 subnet. In the jigs we use in-house, we refer to these traffic servers using the prefix `av` followed by the last octet of the IP address. For example, the traffic server whose IP address is 192.168.150.52 is `av54`.

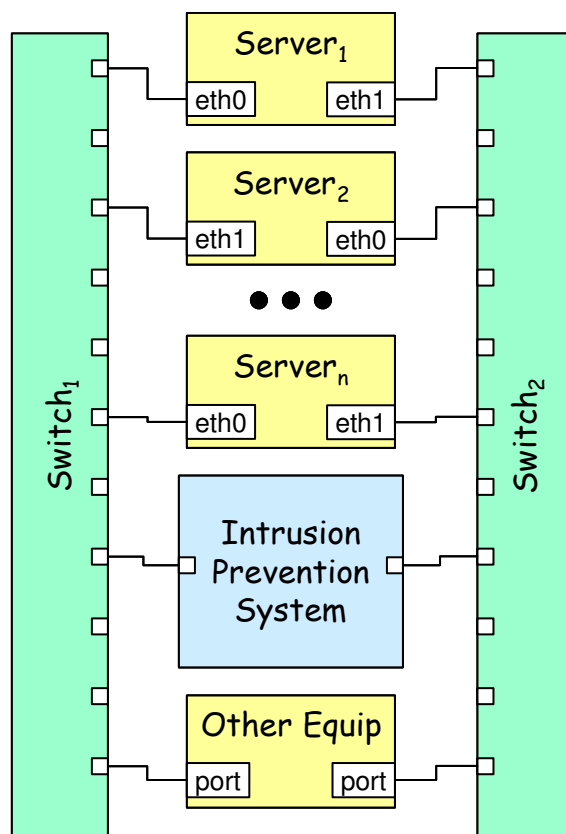


Figure 2: The Tomahawk test jig.

Each traffic server runs a copy of Tomahawk. Traffic from multiple servers is aggregated through a switch before passing through the NIPS. Other test equipment, such as a SmartBits, Agilent, or WebAvalanche can similarly be aggregated through the switches as shown in the figure.

The Ethernet interfaces of the traffic servers are connected to the switches in alternating pairs. For instance, eth0 on av53 is connected to switch₁, but eth0 on av54 is connected to switch₂. The interfaces continue to alternate in this fashion for all servers in the jig.

This setup allows tests that evaluate the performance impact of the NIPS without rewiring the Tomahawk Jig. For example, if a program on av54 contacts a server at 192.168.151.51, traffic will be routed through eth1 on av54, switch₁, the NIPS, switch₂, and eth1 on av53. If the same program on av59 contacts a server at 192.168.151.51, traffic will be routed through eth1 on av59, switch₂, and eth1 on av53. The impact of the NIPS on network performance can be evaluated by measuring the difference in the amount of time needed to complete a task such as copying a file in the two configurations.

The Tomahawk Jig can also be used without modification to run attacks through evasion tools such as fragroute [6]², as explained in Appendix B.

Aggregate throughput is measured by gathering statistics over the management network. Each traffic server runs a small program, called `qatcld.tcl`, which is used to gather statistics from the traffic servers. These statistics include the number of bytes and packets sent and received on eth0 and eth1. A master controller computes and displays the total amount of traffic passing through the system, not including traffic generated by the “other equipment” in the jig.

The Tomahawk Jig is quite flexible in its use. For example, servers can perform multiple tasks with Tomahawk: some traffic servers can replay attacks using Tomahawk, other servers can generate background load using Tomahawk, while other servers can check application latency (such as timing how long it takes to copy a large file or fetch a web page through the NIPS). The SmartBits can get an accurate measurement of network latency while all of these actions perform on the servers.

Appendix B describes the software installation process for the Tomahawk Jig, followed by step-by-step instructions for conducting the tests. The next section shows how to use the Tomahawk Jig to test network and application performance.

Networking and Application Performance Testing

Although there are many possible ways to quantify the performance of a network device, experience has shown that a few measurements for characterizing NIPS performance are invaluable. These are:

- *Throughput* is the total bandwidth, in megabits per second (Mbps) that can pass through a NIPS.
- *Network latency* is the amount of time, in microseconds, that it takes for a packet to pass one-way through the NIPS.

² By default, fragroute requires attack traffic to be sent out eth0. Changing this behavior requires significant modifications to the fragroute source code.

- *Application latency* is the time that the NIPS adds to the completion of an application level task. For example, if a file copy takes 10 seconds to complete on a network without a NIPS, and 11 seconds to complete on a network with a NIPS, then the application latency is 1 second or 10%.
- *Connections per second* measures how many TCP connections per second can be cleanly set up and torn down when a NIPS is deployed in a network.

All of these measures can be highly dependent on the traffic mix that is running through the device at the time the measurement is taken. Since traffic mix can vary widely from one network to another, it is worthwhile to spend a little time discussing this aspect of testing.

Traffic Mix

When testing switches, it is usually sufficient to test at extremes and assume that the device will work well in between. For instance, you can perform a common test for switch performance on both the smallest and largest packets accepted by the switch (e.g., 64 and 1518 bytes). If performance is acceptable at these extremes, it is likely to be acceptable in-between.

This strategy works well because only one degree of freedom typically affects switch performance, namely packet size. In contrast, NIPS products are designed to inspect traffic at the application layer (layer 7). The network performance of a NIPS is a function of traffic ordering, payload contents, protocol, and many other factors. With this many variables, enumerating all the possible extremes becomes impractical because the number of combinations grows considerably larger.

As a consequence, TippingPoint chooses to evaluate NIPS performance with a few, well chosen mixes that are representative of different environments and extremes. If a NIPS performs well in all these environments, it is likely to perform well in most environments. An alternative is to gather a packet trace from the target environment for the NIPS and use it in performance testing, in addition to the extremes.

Latency and Throughput

To measure network performance, we use Tomahawk to replay the various traffic mixes and simultaneously measure network latency. Our specific mixes include HTTP, FTP, and UDP traffic. We encourage you to use a packet trace gathered from your own environment. A guide to running the tests using these traces can be found in Appendix B.

A provided tool (`qatool`) contacts the `qatcld` daemon on each traffic server every 2 seconds to collect statistics from the traffic server's NICs. `Qatool` tallies these statistics and computes the aggregate throughput of the NIPS in Mbps.

To measure latency, a network measurement tool such as a SmartBits or Agilent can be used, if available. If such a tool is not available, the Linux utility `ping` may be used. If the latter is used, it is important to run `ping` between two otherwise idle traffic servers, since the processing associated with Tomahawk can significantly affect the measured latency. It is also important to note `ping` measures round trip time; one-way latency will be half the reported value.

Regardless of what tool is used to measure latency, the switches in the test jig can introduce latency. For this reason, you should measure the baseline latency and throughput of the system with the NIPS replaced by a wire. This procedure also verifies that everything in the jig is working correctly.

For example, suppose the baseline ping time is 200 microseconds and the baseline throughput (reported by `qatool`) is 3 Gbps. When the NIPS is placed inline, the throughput drops to 1.36 Gbps and the ping time increases to 420 microseconds. This data indicates that the aggregate throughput of the NIPS is 1.36 Gbps and the one-way latency is $(420-200)/2 = 110$ microseconds.

Connections per Second

Tomahawk can be used to evaluate the maximum connection setup rate of a NIPS. To perform this test, we created a script that opened, then immediately closed, a connection to a web server 1000 times. We used `tcpdump` to capture the traffic generated by this script. The result is a PCAP that contains 1000 TCP connections, each with a full 3-way handshake (SYN, SYN-ACK, ACK) and 3-way shutdown (FIN-ACK, FIN-ACK, ACK).

We use Tomahawk to replay this PCAP 250 times on each of the traffic servers in the test jig. The result is 250,000 connections per traffic server. When played through a switch, each traffic server completes its run in about 7 seconds, which corresponds to about 35,700 connections/second. With 3 traffic servers, 750,000 unique connections are created at a rate of 107,100 connections/second. Each traffic server uses a different IP address range to ensure that all 1 million TCP 4-tuple are unique during the course of a run.

If the NIPS is placed inline, completing the replay of all 1 million connections may take longer. For instance, when an IntruShield 2600 is placed inline, the test with 1 traffic server takes 56.8 seconds to complete (a connection rate of 4405 connections/second). With 2 traffic servers it takes 106 seconds to complete (a connection rate of 4716 connections/second).

Note: A guide to running the tests using the test jig and included software can be found in Appendix B.

Recursive Web Retrieval

Our third performance test measures the impact of the NIPS on applications, in particular, web browsing. NIPS are often deployed to protect web servers. This test measures the performance impact of such a NIPS on clients browsing a protected server. Our test uses the Linux tool `wget` to recursively retrieve many files from an Apache web server. As a baseline, we measure how long it takes to retrieve the files through the switch alone. We repeat the process, retrieving through the switch plus the NIPS. The relative difference is the impact of the NIPS. For example, if the baseline retrieval takes 10 seconds and the NIPS retrieval takes 12 seconds, the NIPS has a 20% impact on performance ($(12-10)/10 = 20\%$).

In the test jig (figure 2), `server1` runs the web server. To measure the baseline, we execute the client (`wget`) on `server3`. We then run the same client on `server2` to measure the impact of the NIPS. The Linux utility `time` is used to measure performance. Each measurement is taken three times, and the median measurement is used.

Note: A guide to running the tests using the test jig and the included software can be found in Appendix B.

NFS File Copy

NIPS are often deployed in internal LANs to segment the network into security zones. Our fourth performance test measures the impact of the NIPS on LANs by measuring its impact on network file system operations.

As with the recursive web retrieval test, we measure the wall clock time needed to complete a file copy operation with and without the NIPS. The relative difference is the impact of the NIPS.

In the test jig (figure 2), server₁ runs the NFS server. The exported file system is mount on server₂ and server₃. We execute the command `time cat filename > /dev/null` on both server₂ and server₃ three times, unmounting and remounting the file system between measurements to ensure that no part of the file is cached locally.

As before, the difference in the execution time with and without the NIPS inline is the performance impact of NIPS.

Note: A guide to running the tests using the test jig and the included software can be found in Appendix B.

Security Performance Testing

Evaluating the security performance of a NIPS consists of two parts: verifying that the NIPS can accurately and reliably detect and block attacks, and verifying that the NIPS does not block legitimate traffic.

Vendors often confuse the issue by highlighting their different mechanisms for detecting attacks. These mechanisms include signatures, protocol anomaly, application anomaly, and behavioral or statistical anomaly. Regardless of how the NIPS detects the attack, the goal of a NIPS is simple: to block the attack. We use the term *filter* to denote the attack detection function, independent of the detection mechanism.

There are many considerations when evaluating a NIPS's filter set, including:

- *Attack Recognition* is what attacks the NIPS purportedly detects and blocks.
- *Repeatability* is whether the NIPS can block the attack each and every time the attack is sent through the NIPS.
- *Evasion Resistance* measures the NIPS ability to detect attacks when an attacker attempts to modify the attack to make it more difficult for an IDS/IPS to detect. Evasions include simple changes to an exploit and the use of automated IDS evasion tools such as fragroute [2] and whisker [7].

Security Performance: Attack Recognition

Experimentally determining the attack recognition capabilities of a NIPS is difficult. Since there are tens of thousands of known vulnerabilities and an infinite number of attacks, a sampling methodology must be used where a modest number of attacks are played through the NIPS. A typical test runs anywhere from 10 to 100 randomly selected attacks through the NIPS, with the assumption that, if the vendor provides good coverage for those attacks, they are likely to cover other attacks well. The problem is that a good, random sample of attacks is hard to find.

Commercial tools such as Blade IDS Informer [4] or scanning tools such as Nessus [5] are often used to supply the attacks. Unfortunately, using these tools to evaluate NIPS coverage gives misleading results. Scanning tools like Nessus are designed to detect systems that might be vulnerable to attack. The technology uses “banner scraping” to determine the version of software running on the server being scanned. For example, such a tool might request a web page and determine from the HTTP response header that the server is running IIS 5.0, which is known to be vulnerable to a certain attack. The scanner would then report that the server is vulnerable. Since no attack has been launched, it would be wrong for the NIPS to block the traffic (blocking it would mean blocking legitimate traffic).

Commercial tools that launch known attacks are also of limited value because the vendors have access to the attacks and can ensure that their filter set detects every attack. Consider the following scenario. NIPS vendors X and Y run 100 attacks from Blade in the lab. Vendor X discovers that their product misses 50 application anomaly attacks. In response, vendor X creates 50 signature-based filters for the missed attacks³. Meanwhile, vendor Y’s product only misses 2, and in analyzing the missed attacks, Y finds that they are invalid.

A customer now repeats the same test for X and Y in the customer lab and finds that X outperforms Y: X blocks 100% of the attacks, Y blocks 98%. The customer erroneously concludes that X is better than Y, not knowing that the 2 missed attacks are invalid.

So how can attack coverage be evaluated? The NSS labs [8] evaluated each product using a secret set of 120 attacks that were unknown to the vendors before the test. After running the test, each vendor is given a chance to respond to the missed attacks, either proving that they are invalid or adding coverage. If the attack is found to be invalid, it is counted as a false positive to the vendors that (incorrectly) detect the attack. This methodology is quite good, provided enough valid, important attacks can be gathered to get a statistically significant sample. Since gathering those attacks is a tremendous amount of work, a second method is to simply analyze the coverage of the filter set by comparing it against a list of significant threats, such as those published by SANS [9].

Despite these constraints, customers will want to see a NIPS in action, blocking attacks. The software included with this document contains 16 different attacks that can be used to test whether a NIPS can block attacks. The software uses Tomahawk to replay network traffic between two network cards on the same machine (eth0 and eth1 in figure 1). The packets arrive at the NIPS interfaces in the same order as they would in the live network. If the NIPS

³ Signature-based filters use a pattern from the traffic to detect the attack, and are the weakest form of protection.

blocks a packet, Tomahawk resends the packet up to a configure number of times (typically 10 retries).

Note: A guide to running the tests using the test jig and the included software can be found in Appendix B.

This methodology provides independent verification that the NIPS blocks the attack. It can also be used to replay to same attack many thousands of times, as discussed in the next section.

Security Performance: Repeatability

Modern network attacks can generate significant load on a NIPS and a network. It is important that a NIPS block an attack each and every time an attack is seen, regardless of the background traffic. We use *repeatability* testing to check this functionality.

The idea of repeatability testing is simple: replay a set of attacks a large number of times at high speed. If a NIPS blocks the attack once, it should block it every time. If a NIPS misses an attack once, it should miss it every time.

An important detail is that each attack should appear to come from a different host. Many NIPS block a flow once an attack is identified within the flow. Flows are identified by IP source and destination addresses and TCP/UDP port numbers (the so-called “host-port quadruple”). If the same quadruple is used in multiple attacks, the repeatability test is weakened considerably because once the NIPS correctly identifies the attack in the flow, it will drop every packet associated with that quadruple. This can give the appearance that the NIPS is correctly identifying each subsequent attack without testing the NIPS capabilities to do so.

The Tomahawk toolset can replay the 16 provided attacks 1000 times in 32 seconds over a bare wire (about 500 attacks/second for each traffic server). Each attack is given a unique host/port quadruple, avoiding the problem described above.

Note: A guide to running the tests using the test jig and the included software can be found in Appendix B.

A passing grade should be given to NIPS that exhibits perfect repeatability.

Security Performance: Evasion resistance

Our final test checks the NIPS ability to resist evasions. Hackers modify their attack to increase the difficulty of IDS/IPS detection. Automated IDS evasion tools such as fragroute generate a particularly important class of evasions.

Fragroute is designed to exploit ambiguities in the IP and TCP protocols. To use it, you run the command

```
fragroute -f script target
```

where *script* is a text file describing the evasions to use, and *target* is the IP address of the victim under attack. Fragroute modifies the kernel routing table so that all traffic destined for the *target* is passed through fragroute, where the evasions specified in *script* are applied.

Once fragroute is installed, an attack can be launched at the victim. Our attack is based on HTTP, and fetches a URL used by the Nimda worm [10]. It is important that a NIPS correctly block the attack, independent of the script used. It is not very important that a NIPS correctly identify the attack once it has been subject to evasion, since the evasion itself does not occur naturally and is therefore reason enough to block the traffic.

A passing grade should be given to NIPS that can detect and the valid evasions that can be generated by fragroute. It is possible to use fragroute to generate traffic that is sufficiently modified that the intended victim cannot decode it. In this case, the attack could never succeed. For this reason, it is important to verify that any homegrown evasions allow the attack to continue to work. The accompanying software includes 12 evasions that are known to work.

Note: A guide to running the tests using the test jig and the included software can be found in Appendix B.

Conclusions

The tests above can be combined to test different aspects of the NIPS simultaneously. For example:

Blocking under load.

The throughput and repeatability tests can be run simultaneously. The NIPS should be able to maintain high repeatability even with background traffic running.

Network performance while blocking.

The application performance tests (NFS and HTTP) can be run on a pair of servers while a third runs the repeatability tests to generate attacks. Given a moderate attack load (e.g., 500 attacks/second), the performance of the applications should be relatively unaffected.

Manageability while under attack.

It is important that a NIPS remain manageable while the network is under attack. You can use the repeatability test on multiple servers to generate a large number of attacks, and then try to manage the NIPS using the management system.

Testing NIPS is a complex and challenging problem involving both network and security performance testing. The tools and methodologies described in this paper represent a large step forward in the creation of fair criteria for evaluating NIPS products. Despite this, more research and development needs to be done to improve the methodologies and tools. In particular, the problem of fairly evaluating attack coverage remains an open issue.

Appendix A. The Engineering of Tomahawk

Tomahawk simulates a network attack by sending data in one or more packet traces from the attacker NIC to the victim NIC (and vice versa). A packet trace (PCAP) is captured in tcpdump format.

When Tomahawk loads a PCAP, it assigns the packet trace to one of two categories: those generated by the “attacker” and those generated by the “victim”. When Tomahawk replays the attack, it sends attacker packets on the attacker NIC and victim packets on the victim NIC. The tool effectively produces the effect of a piece of network equipment in the path between the attacker and victim NICs (e.g., a NIPS) sees the same traffic, on the same interfaces, as it would have seen had it been installed in the network at the same time and place as the sniffer when the trace was captured.

The following processing occurs when Tomahawk loads a PCAP. The first time an IP address is seen in a PCAP file, Tomahawk assigns the IP address to the attacker if it is in the IP source address field of the packet, or to the victim if it is in the destination field. For instance, consider a PCAP consisting of a standard three-way TCP handshake contains 3 packets:

Packet 1 (SYN): ip.src = 192.168.2.5 ip.dest = 192.168.2.4

Packet 2 (SYN-ACK): ip.src = 192.168.2.4 ip.dest = 192.168.2.5

Packet 3 (ACK): ip.src = 192.168.2.5 ip.dest = 192.168.2.4

When Tomahawk reads the first packet, the address 192.168.2.5 is encountered for the first time in the source field and the address 192.168.2.4 in the destination field. Tomahawk assigns the address 192.168.2.5 to the attacker and the address 192.168.2.4 to the victim.

To replay the sequence above, Tomahawk begins by sending packet 1 from the attacker NIC. When this packet arrives on the victim NIC, Tomahawk sends packet 2 out the victim NIC and waits for packet 3 to arrive on the attacker NIC. When the packet arrives, Tomahawk sends packet 3 on the attacker NIC. When the last packet arrives on the victim NIC, Tomahawk outputs that it has completed the PCAP.

If a packet does not arrive after a specified timeout (typically 100 milliseconds), Tomahawk infers that the packet is lost and retransmits the packet. For example, if Tomahawk sends packet 2 on the victim NIC and does not receive it on the attacker NIC within the timeout, it resends packet 2. If the packet does not arrive after a specified number of retransmissions (typically 5), the replay of that PCAP is aborted and Tomahawk outputs a message indicating that the session has timed out.

Note: When attack traffic plays through a NIPS, if Tomahawk reports that the PCAP containing the attack has timed out, the IPS has correctly blocked the attack. If Tomahawk reports that the PCAP has completed, the IPS missed the attack, regardless of what any NIPS log indicates. In this way, Tomahawk is an independent auditor of the NIPS functionality.

To ensure that the packet is correctly routed through the switches, the Ethernet MAC addresses are rewritten when the packet is sent. In addition, Tomahawk also rewrites the IP addresses and updates the packet's checksums accordingly. For instance, in the trace above, when Tomahawk sends packet 1, the IP source address of the packet that appears on the wire is 10.0.0.1, and the IP destination address is 10.0.0.2. This feature allows Tomahawk to replay the same PCAP multiple times in parallel. If two instances of the PCAP above were replayed in parallel, the IP addresses in the first instance would be 10.0.0.1 and 10.0.0.2, while the IP addresses in the second instance would be 10.0.0.3 and 10.0.0.4, allowing the IPS to treat each session independently.

By rewriting IP addresses, Tomahawk can replay hundreds or even thousands of copies in parallel, each with a different IP address. In this way, Tomahawk can simulate activity on a network with tens of thousands of machines. Since Tomahawk loads a PCAP into memory before replay, it can comfortably generate several hundred Mbps of traffic with conventional PCs.

Appendix B. A Guide to the Accompanying Software

Overview

The software that accompanies the Tomahawk evaluation tool supports the test jig shown in figure 1. Each traffic server machine is assumed to meet the following minimum requirements:

- Pentium with 1GHz or faster processor
- 512 MB of memory
- Red Hat Linux 7.2 or later (or equivalent Linux distribution)
- 2 gigabit Network Interfaces Cards (NICs) for data, assigned to eth0 and eth1. The Intel ProLAN adapters are inexpensive and work great.
- 1 NIC for management, assigned to eth2

On each traffic server, eth0 interfaces are assigned IP addresses out of the 192.168.150.0/24 subnet, while eth1 interfaces are assigned IP addresses out of the 192.168.151.0/24 subnet. The management IP can be assigned any address. For purposes of this documentation, we assume the management IP addresses of the 3 traffic server machines are in the range 192.168.0.51 – 192.168.0.53.

In addition, a management workstation is required to run the traffic control software. This software controls the traffic servers and gathers statistics to report throughput. Ideally, this is a separate machine so that processing on the management workstation does not interfere with processing on the traffic servers, but the traffic control software can also be run on a traffic server in a pinch.

Installation

In the instructions that follow, the prompt % indicates a command should be executed on every traffic server. Otherwise, the prompt av5n% indicates that a command should be run on traffic server *n*. For example, the prompt av53% indicates that a command should be executed only on traffic server 1.

Step 1

Copy the file qa.tgz and pcaps.tgz to /usr/local on the target system. Create /usr/local/qa using the following sequence of commands:

```
cd /usr/local
tar xvfz qa.tgz
tar xvfz pcaps.tgz
```

Step 2

Configure networking by editing, then copying the files in /usr/local/qa/install/etc to /etc/sysconfig/network-scripts/. Adjust the IPADDR variable in the other files to identify the server. By convention, the IP address of eth0 on server *n* is 192.168.150.5*n* and the IP address of eth1 on server *n* is 192.168.151.5*n*. For example, the IP address of eth0 on server 1 is 192.168.150.51, and the IP address of eth1 on server 3 is 192.168.151.53. See figures 1. The default configuration is to use DHCP for the management port.

```
cd /usr/local/qa/install/etc
<edit ifcfg-eth*>
cat ifcfg-eth0 > /etc/sysconfig/network-scripts/ifcfg-eth0
cat ifcfg-eth1 > /etc/sysconfig/network-scripts/ifcfg-eth1
cat ifcfg-eth2 > /etc/sysconfig/network-scripts/ifcfg-eth2
```

We use `cat` to overwrite the existing files, rather than `cp`, because the files have multiple hard links. `cp` breaks those links, `cat` does not. As a convenience, the script `/usr/local/qa/install/etc/install.sh` has the `cat` commands builtin.

Alternatively, you can use `dhcp` to obtain the management IP by copying the file `ifcfg-eth2.dhcp` to `/etc/sysconfig/network-scripts/ifcfg-eth2`:

```
cat ifcfg-eth2.dhcp > /etc/sysconfig/network-scripts/ifcfg-eth2
```

Restart the interfaces to force the changes to take effect:

```
ifdown eth0
ifdown eth2
ifdown eth1
ifup eth0
ifup eth2
ifup eth1
```

Step 3

Start the NFS server on `server1`. First, export the root file system for the NFS performance tests. Add the following lines to `/etc/exports` on `server1`:

```
/ 192.168.150.0/24(rw,no_root_squash)
/ 192.168.151.0/24(rw,no_root_squash)
```

Run `exportfs` to get NFS to read the files, and restart the NFS service:

```
av53% exportfs -a
av53% chkconfig nfs on
av53% service nfs restart
```

Step 4

Set up servers 2 and 3 to be NFS clients. Add the following line to `/etc/fstab` on servers 2 and 3:

```
192.168.150.51:/ /av53 nfs rsize=8192,wsiz=8192,timeo=14
```

Create the mount point for NFS on servers 2 and 3:

```
av54% mkdir /av53
av59% mkdir /av53
```

Lastly, check to see that you can mount the NFS server on the clients

```
av54% mount /av53
av59% mount /av53
```

Step 5

Set up the web server on server₁. Start by locating the directory where HTML content is stored using the following command:

```
av53% grep ^DocumentRoot /etc/httpd/conf/httpd.conf
DocumentRoot "/var/www/html"
```

In this example, the target directory is `/var/www/html`. Now, untar the files in `/usr/local/qa/install/www.tgz` into this directory:

```
av53% cd /var/www/html
av53% tar xvfz /usr/local/qa/install/www.tgz
```

Start the web server:

```
av53% chkconfig httpd on
av53% service httpd restart
```

Step 6

On all the servers, install ActiveTcl using the following sequence of commands:

```
av54% cd /usr/local/qa/install/ActiveTcl8.3.4.2-linux-ix86
av54% ./install.sh
```

Follow the defaults in the prompts.

Step 7

Now you need to set up your environment variables. For your convenience, the shell script `/usr/local/qa/install/install.sh` will modify `$HOME/.bashrc` as described in the following steps:

On all the servers, add `/usr/local/ActiveTcl/bin` and `/usr/local/qa/bin` to your PATH. This is typically done by adding a line such as one of the following to `$HOME/.bashrc`, `$HOME/.profile`, or `$HOME/.cshrc`.

For a csh or similar shell:

```
set PATH "/usr/local/ActiveTcl/bin:$PATH:/usr/local/qa/bin"
```

For a sh or similar shell:

```
PATH="/usr/local/ActiveTcl/bin:$PATH:/usr/local/qa/bin"
export PATH
```

For a bash:

```
export PATH="/usr/local/ActiveTcl/bin:$PATH:/usr/local/qa/bin"
```

On each server, you must set `qatcld` to start at runtime. You can configure this setting by adding the following lines to `/etc/rc.d/rc.local`:

```
touch /var/lock/subsys/local
rm -f /var/run/qatcld
/usr/local/qa/bin/qatcld.sh &
```

You can either reboot the server to have these settings take effect or manually execute the above commands to avoid the reboot.

As a final note, in order to run, the Management station must be running X windows server to display the GUI for measuring the traffic load generated by the servers. Linux systems all include the X server. Cygwin (<http://cygwin.com>) is Windows freeware that includes an X server.

Network Performance Tests: Latency and Throughput

To measure performance, we use Tomahawk on each of the traffic servers to generate the traffic load and a separate tool to monitor the amount traffic generated by each server, along with the total throughput and latency.

Monitoring throughput

To monitor throughput, run the following command on any of the traffic servers

```
av53% qatool.tcl &
```

This command will pop up a window on the management workstation⁴. Press the *Load* button and load the script `perf.txt` in `/usr/local/qa/lib`. Then press the *Run* button.

The tool will begin sampling data from each traffic server and present the results in the summary window. The raw data is taken from `/proc/dev/net`, which the operating system updates about every 2 seconds using data from the device driver. The number you want to watch is in the lower left corner – this is the total throughput of the system.

Measuring latency

Latency can be measured most accurately using a separate device, such as a SmartBits, which can be install in the jib as shown in figure 2. Alternatively, you can ping from one traffic server to another:

```
av54% ping 192.168.150.54
```

Be sure that your ping is going through the NIPS. If you ping from av53 to av54, for example, the traffic will only go through the switch so you will not be measuring NIPS latency.

⁴ Note that the management server must be running and X windows server in order to display the GUI. See the Installation section above for additional details.

As you monitor throughput and latency, you can start generating load. You should start by getting a baseline measurement by putting the NIPS in bypass mode (or replacing it with a wire). This allows you to validate that the jig is working correctly and to measure the offered load and baseline latency introduced by the operating system and the switches. With this established baseline, the NIPS can be put back inline.

Once you have the baseline and system latency, you can calculate the one-way latency of the NIPS. The *baseline latency* is defined as the latency of the test jig when the NIPS is replaced by a wire. The *system latency* is the latency of the test jig with the NIPS running inline, under load. The difference between the system and baseline latency is the latency introduced by the NIPS.

If you are using ping to measure latency, it is important to note that the latency reported by ping is the round trip time, not the one-way latency. The one-way latency through the NIPS is then half the difference between the system and baseline ping. For example, if the baseline ping time is 200 microseconds and the system ping time is 400 microseconds, the one-way NIPS latency is $(400-200)/2 = 100$ microseconds.

Profile 1: Pure TCP – HTTP and FTP

This profile consists of 66% HTTP and 33% FTP traffic. To run it, use the following commands on each of the traffic servers:

```
av53% perf_http
av54% perf_ftp
av59% perf_http
```

You can monitor the throughput on the `qatool` display. `Perf_http` is a short shell script that uses Tomahawk to replay traffic captured off a live network consisting of only HTTP traffic. `Perf_ftp` is similar, except that it replays the download of a large FTP file. This traffic mix is similar to that generated by the WebAvalanche and represents a “best case” scenario for most NIPS. To stop the traffic flow, press the Ctrl-C key in each command shell to kill the various `perf*` scripts.

Profile 2: Pure UDP

This profile consists of 33% HTTP, 33% FTP, and 33% UDP traffic. To run it, use the following commands on each of the traffic servers:

```
av53% perf_http
av54% perf_ftp
av59% perf_udp
```

As before, you can monitor the throughput on the `qatool` display. The UDP traffic used in the trace was captured on a local area network.

Profile 3: Local profile

It is straightforward to modify these scripts to meet your own needs. We recommend that you capture traffic from your own network using `tcpdump`. To do this, connect the `eth0`

port on one of the traffic servers (e.g., av53) to a mirrored port and execute the following command:

```
av53% tcpdump -s 0 -i eth0 -w /usr/local/pcaps/my.pcap
```

Copy the file to the other server:

```
av53% scp /usr/local/pcaps/my.pcap 192.168.0.52:/usr/local/pcaps
av53% scp /usr/local/pcaps/my.pcap 192.168.0.53:/usr/local/pcaps
av53% scp /usr/local/pcaps/my.pcap 192.168.0.54:/usr/local/pcaps
```

Finally, create a copy of `perf_http` in `/usr/local/bin` and edit it to change the name of the PCAP file, then copy it to the other traffic servers:

```
av53% cd /usr/local/bin
av53% cp perf_http my_perf
av53% edit my_perf
av53% scp my_perf 192.168.0.52:/usr/local/bin
av53% scp my_perf 192.168.0.53:/usr/local/bin
av53% scp my_perf 192.168.0.54:/usr/local/bin
```

Since the traffic load that Tomahawk can generate is dependent on the mix of traffic in the pcap (smaller packets = less performance), it's a good idea to first measure the performance with the NIPS in layer-2 fallback (or replaced by a wire) to see what load is being offered to the NIPS.

Network Performance Tests: Connections per Second

To run the connections per second test, execute the following command on a traffic server:

```
av53% cps
time to create 250,000 connections:
real    0m3.268s
user    0m0.040s
sys     0m0.120s
```

It outputs the amount of time it took to replay 250,000 connections. In this example, it took 3.268 seconds, which corresponds to $250,000/3.268 = 76,500$ connections/sec. If you need to generate higher loads, you can do so by running it in parallel on multiple traffic servers. For example, running it on all four traffic servers, it takes the slowest server 5 seconds to complete. The four traffic servers created a total of one million connections, so the connection/second rate is $1,000,000/5 = 200,000$ connections/second.

Application Performance Tests: Recursive Web Retrieval

On av53, make sure the web server is running:

```
av53% service httpd restart
```

On av54, execute the baseline test three times (only one output is shown below). Since this traffic does not go through the NIPS, it represents the fastest the system will allow:

```
av54% time wget -r -o /dev/null http://192.168.150.51/
real    0m3.101s
user    0m0.130s
sys     0m0.070s
```

Execute the same test three times on av59, so that the traffic must flow through the NIPS (only one output is shown below):

```
av59% time wget -r -o /dev/null http://192.168.150.51/
real    0m3.499s
user    0m0.130s
sys     0m0.070s
```

Finally, compute the performance impact as a percentage, using the median value: $(3.499 - 3.101) / 3.101 = 0.128 = 12.8\%$.

Application Performance Tests: NFS File Copy

On av53, make sure the NFS server is started:

```
av53% service nfs restart
Shutting down NFS mountd:           [ OK ]
Shutting down NFS daemon:          [ OK ]
Shutting down NFS quotas:          [ OK ]
Shutting down NFS services:        [ OK ]
Starting NFS services:              [ OK ]
Starting NFS quotas:                [ OK ]
Starting NFS daemon:                [ OK ]
Starting NFS mountd:                [ OK ]
```

On av54, execute the baseline test three times (only one output is shown below):

```
av54% mount /av53

av54% time cat /av53/usr/local/pcaps/ftp233M.tcpdump > /dev/null
real    0m4.456s
user    0m0.000s
sys     0m0.030s

av54% umount /av53
```

Execute the system test three times on av59 (only one output is shown below):

```
av59% mount /av53

av59% time cat /av53/usr/local/pcaps/ftp233M.tcpdump > /dev/null
real    0m4.552s
user    0m0.000s
sys     0m0.030s

av59% umount /av53
```

Using the median from both measurements, compute the performance impact: $(4.552 - 4.456) / 4.456 = 0.022 = 2.2\%$.

Security Performance: Attack Recognition

To run this test, place the NIPS in block mode. Execute the script `/usr/local/qa/bin/attacks` on `av53`:

```
av53% attacks
```

This runs 16 fairly basic attacks through the NIPS and reports whether or not the attacks were blocked.

Security Performance: Repeatability

To run this test, place the NIPS in block mode. Execute the script `/usr/local/qa/bin/attacks` on `av53`, passing the number of times to play each attack. For example, to play each attack 1000 times (for a total of 16000 attacks), you would use:

```
av53% attacks 1000
```

This test checks whether the NIPS can reliably block attacks. If some of the attacks are blocked and some permitted, then the NIPS is either letting some leak through or blocking traffic it shouldn't be blocking, depending on whether or not the NIPS has a filter for the attack.

As usual, you can verify that the jig is functioning correctly by replacing the NIPS with a wire (all attacks should then be permitted).

Note that you can also generate background traffic with the other traffic servers while running attacks on `av53`.

Security Performance: Evasion Resistance

Start the Apache web server on `av53`:

```
av53% service httpd restart
```

Verify the server is running using the following command on `av54`:

```
av54% wget http://192.168.150.51/
```

Also verify that the route to `192.168.150.51` is via `eth0`:

```
av54% route -n | grep 192.168.150
192.168.150.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
```

Verify that the NIPS will detect and block the following attack:

```
av54% wget 'http://192.168.150.51/c/winnt/system32/cmd.exe?/c+dir'
```

This URL is one of the Nimda attack vectors, and any reliable and effective NIPS should be able to block it. The `wget` command will hang if the NIPS successfully blocks the attack. You may need to tweak the NIPS configuration and repeat to get it to block the attack.

At this point, we know that the NIPS can block the attack. Now check how it does with evasions [1]. Start up fragroute using the `tcp_frag-win32` script:

```
av54% cd /usr/local/qa/bin/scripts
av54% ping -c 1 192.168.150.51
av54% fragroute -f tcp_frag-win32 192.168.150.51
```

This script uses the overlapping TCP segment evasion from section 5.4.2 of Ptacek's classic paper on IDS evasion.

Verify that the client can contact the server and that NIPS still blocks the attack if it is run through fragroute (you will need to do this in another window):

```
av54% wget http://192.168.150.51/
av54% wget 'http://192.168.150.51/c/winnt/system32/cmd.exe?/c+dir'
```

In the first window, kill and restart fragroute using the `tcp_frag-unix` script:

```
av54% cd /usr/local/qa/bin/scripts
av54% ping -c 1 192.168.150.51
av54% fragroute -f tcp_frag-unix 192.168.150.51
```

Try it again:

```
av54% wget http://192.168.150.51/
av54% wget 'http://192.168.150.51/c/winnt/system32/cmd.exe?/c+dir'
```

You can repeat this process with the different fragroute scripts in the directory `/usr/local/qa/bin/scripts`. A reliable and effective IPS detects and blocks all the attacks.

References

- [1] Thomas H. Ptacek, Timothy N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection," January 1998, Secure Networks Inc., http://www.insecure.org/stf/secnet_ids/secnet_ids.html
- [2] Richard Stevens, "TCP/IP Illustrated, Volume 1: The Protocols," Addison-Wesley, 1994, ISBN 0-201-63346-9
- [3] John K. Ousterhout, "Tcl and the Tk Toolkit", Addison-Wesley, 1994, ISBN 0-201-63337-X
- [4] <http://www.blade-software.com/IDSInformer.htm>
- [5] <http://www.nessus.org/>
- [6] <http://monkey.org/~dugsong/fragroute>
- [7] <http://www.ussrback.com/docs/papers/IDS/whiskerids.html>
- [8] <http://www.nss.co.uk/>
- [9] "@RISK: The Consensus Security Alert", The SANS Institute, Bethesda, Maryland, <http://www.sans.org/newsletters/risk/>
- [10] A. Mackie, J. Roculan, R. Russell, M. Van Velzen, "Nimda Worm Analysis", <http://aris.securityfocus.com/alerts/nimda/010919-Analysis-Nimda.pdf>