NFR Sentivist Smart Sensor 100C V1.3

Technical Evaluation

An NSS Group Report

First published June 2005 (Version 1.0)

Published by The NSS Group
Security Testing Laboratories
Mas la Carrière, Route de Ganges
30440 Sumène, France

Tel : +33 (0)4 67 81 49 11
E-mail : info@nss.co.uk
Internet : http://www.nss.co.uk

TABLE OF CONTENTS

TABLE OF FIGURES

The NSS Group

The NSS Group is the world's foremost independent security testing facility.

With British headquarters, and security and network infrastructure testing facilities in the South of France, The NSS Group offers a range of specialist IT, networking and security-related services to vendors and end-user organisations world-wide.

The NSS Group's Security Testing Laboratories are available to vendors and end-users for fully independent testing of networking, communications and security hardware and software.

The NSS Group also operates certification schemes for vendors and certification bodies, and currently provides evaluation and certification of a wide range of security products, including IDS/IPS appliances, firewalls, VPNs, Web Application firewalls, multi-function security appliances, cryptographic devices and PKI products.

Output from the labs, including detailed research reports, articles and white papers on the latest network and security technologies, are made available on the NSS web site at http://www.nss.co.uk.

The NSS Group awards are recognised world-wide as being the most desirable and essential when it comes to security products. Vendors consider the awards to be a crucial step in any security-related marketing campaign, whilst feedback from readers of the reports indicates that participation in an NSS Group test and/or one of the **NSS Approved** awards is a prerequisite for any security product in order to be considered for purchase.

Foreword

Following the huge success of the first comprehensive *Intrusion Prevention System* (IPS) test of its kind, The NSS Group is pleased to present the results of its third IPS Group Test, the largest so far, which includes a number of new products not included in the first two reports.

As with the first two Editions, this exhaustive review will give readers a complete perspective of the capabilities, maturity and suitability for immediate deployment of each of the products tested. The NSS Group established this test as IPS products are being actively deployed as a new layer in defence-in-depth security architectures.

The NSS IPS Group Test evaluates the performance, reliability, security effectiveness, and usability of Network IPS products. The test consists of seven sections within three primary areas: *performance and reliability, security accuracy,* and *usability*.

Overall, the brand new test suite contains over **800 individual tests**, many of which are run multiple times, to provide the most thorough and complete evaluation of IPS products available anywhere today. The NSS Group has developed advanced testing methodologies for both *Rate-Based IPS* and *Content-Based IPS* products, since these devices are often very different in operation, although all products tested in this edition of the report are content-based.

It is worth pointing out that not every product submitted for testing receives an *NSS Approved* award. Standards are very high, and only those appearing in this report have received ***NSS Approved*** awards. For this latest edition, **ten** vendors submitted a total of **twelve** products for testing, and **eight** of these passed our stringent testing to receive **NSS Approved**. It is heartening to note that this is a much-improved success ratio over Edition 2.

We believe that our IPS test methodologies - which have been updated again for this test - will become the *de facto* standard for testing in-line Intrusion Prevention/Attack Mitigation devices, and the *NSS Approved* logo an essential item on the list of requirements when purchasing these products.

We also believe that this report is essential reading for anyone considering deploying Intrusion Prevention Systems in their networks, either in a test or live situation, and we hope that you find it both informative and useful in making your purchasing decisions. The latest **IPS Group Test** report can be viewed on-line at *www.nss.co.uk/ips*

Bob Walder

INTRODUCTION

In a survey commissioned by VanDyke Software, some 66 per cent of the companies who responded said that they perceive system penetration to be the largest threat to their enterprises.

The survey revealed that the top eight threats experienced by those surveyed were *viruses* (78 per cent of respondents), *system penetration* (50 per cent), *DoS* (40 per cent), *insider abuse* (29 per cent), *spoofing* (28 per cent), *data/network sabotage* (20 per cent), and *unauthorised insider access* (16 per cent).

Although 86 per cent of respondents use firewalls (a disturbingly **low** figure in this day and age, to be honest!), it is apparent that firewalls are not always effective against many intrusion attempts. The average firewall is designed to deny clearly suspicious traffic - such as an attempt to telnet to a device when corporate security policy forbids telnet access completely - but is also designed to allow some traffic through - Web traffic to an internal Web server, for example.

The problem is, that many exploits attempt to take advantage of weaknesses in the very protocols that **are** allowed through our perimeter firewalls, and once the Web server has been compromised, this can often be used as a springboard to launch additional attacks on other internal servers. Once a “rootkit” or “back door” has been installed on a server, the hacker has ensured that he will have unfettered access to that machine at any point in the future.

Firewalls are also typically employed only at the network perimeter. However, many attacks, intentional or otherwise, are launched from within an organisation. Virtual private networks, laptops, and wireless networks all provide access to the internal network that often bypasses the firewall. Intrusion detection systems may be effective at detecting suspicious activity, but do not provide *protection* against attacks. Recent worms such as Slammer and Blaster have such fast propagation speeds that by the time an alert is generated, the damage is done and spreading fast.

Intrusion Prevention Systems (IPS)

The inadequacies inherent in current defences has driven the development of a new breed of security products known as *Intrusion Prevention Systems* (IPS). This is a term which has provoked some controversy in the industry since some firewall and IDS vendors think it has been “hijacked” and used as a marketing term rather than as a description for any kind of new technology.

Whilst it is true that firewalls, routers, IDS devices and even AV gateways all have intrusion prevention technology included in some form, we believe that there are sufficient grounds to create a new market sector for true *Intrusion Prevention Systems*.

These systems are proactive defence mechanisms designed to detect malicious packets within normal network traffic (something that the current breed of firewalls do not actually do, for example) and stop intrusions dead, blocking the offending traffic automatically before it does any damage rather than simply raising an alert as, or after, the malicious payload has been delivered.

Within the IPS market place, there are two main categories of product: *Host IPS* and *Network IPS*, with the latter being further sub-divided into *Content-Based* and *Rate-Based* (or *Attack Mitigation*) systems.

Host IPS (HIPS)

As with Host IDS systems, the Host IPS relies on agents installed directly on the system being protected. It binds closely with the operating system kernel and services, monitoring and intercepting system calls to the kernel or APIs in order to prevent attacks as well as log them.

It may also monitor data streams and the environment specific to a particular application (file locations and Registry settings for a Web server, for example) in order to protect that application from generic attacks for which no “signature” yet exists.

One potential disadvantage with this approach is that, given the necessarily tight integration with the host operating system, future OS upgrades could cause problems.

Since a Host IPS agent intercepts all requests to the system it protects, it has certain prerequisites - it must be very reliable, must not negatively impact performance, and must not block legitimate traffic. Any HIPS that does not meet these minimum requirements should never be installed in a host, no matter how effectively it blocks attacks.

Network IPS (NIPS)

The Network IPS combines features of a standard IDS, an IPS and a firewall, and is sometimes known as an *In-line IDS* or *Gateway IDS (GIDS)*. The next-generation firewall - the *deep inspection firewall* - also exhibits a similar feature set, though we do not believe that the deep inspection firewall is ready for mainstream deployment just yet.

As with a typical firewall, the NIPS has at least two network interfaces, one designated as *internal* and one as *external*. As packets appear at either interface they are passed to the detection engine, at which point the IPS device functions much as any IDS would in determining whether or not the packet being examined poses a threat.

However, if it should detect malicious traffic, in addition to raising an alert, it will discard the packet(s) and mark that flow as bad. As the remaining packets that make up that particular TCP session arrive at the IPS device, they are discarded immediately.

Legitimate packets are passed through to the second interface and on to their intended destination. A useful side effect of some NIPS products is that as a matter of course - in fact as part of the initial detection process - they will provide “*packet scrubbing*” functionality to remove protocol inconsistencies resulting from varying interpretations of the TCP/IP specification (or intentional packet manipulation).

Thus any fragmented packets, out-of-order packets, or packets with overlapping IP fragments will be re-ordered and “cleaned up” before being passed to the destination host, and illegal packets can be dropped completely.

One thing to watch out for - don't let the "reactive" IDS vendors kid you into believing that they have *intrusion prevention* capabilities just because they can send TCP reset commands or re-configure a firewall when they detect an attack (a worrying piece of FUD that we have noticed in some IDS marketing literature recently).

The problem here is that unless the attacker is operating on a 2400 baud modem, the likelihood is that by the time the IDS has detected the offending packet, raised an alert, and transmitted the TCP Resets - and especially by the time the two ends of the connection have received the Reset packets and acted on them (or the firewall or router has had time to activate new rules to block the remainder of the flow) - the payload of the exploit has long since been delivered….. *game over*! Our guess is that there are not many crackers using 2400 baud modems these days….

A true IPS device, however, is sitting in-line - **all** the packets have to pass through it. Therefore, as soon as a suspicious packet has been detected - and **before** it is passed to the internal interface and on to the protected network, it can be dropped. Not only that, but now that flow has been flagged as suspicious, **all** subsequent packets that are part of that session can also be dropped with very little additional processing. Oh, and for good measure, some products are also capable of sending *TCP Resets* or *ICMP Unreachable* messages to the attacking host.

Rate-Based IPS (Attack Mitigator)

Most NIPS products are basically IDS engines that operate in-line, and are thus dependent on protocol analysis or signature matching to recognise malicious content within individual packets (or across groups of packets). These can be classed as *Content-Based IPS* systems.

There is, however, a second breed of Network IPS that ignores packet content almost completely, instead monitoring for anomalies in network traffic that might characterise a flood attempt, scan attempt, and so on. These devices are capable of monitoring traffic flows in order to determine what is considered "normal", and applying various techniques to determine when that traffic deviates from normal. This is not always as simple as watching for high-volumes of a specific type of traffic in a short space of time, since they must also be capable of detecting "stealth "attacks, such as low-rate connection floods and slow port scan attempts.

Since these devices are concerned more with anomalies in traffic flow than packet contents, they are classed as *Rate-Based IPS* systems - and are also known as *Attack Mitigators*, as they are so effective against DOS and DDOS attacks.

Detection Methods

At one time, most Network IDS/IPS products based their alerts purely on pattern matching packet contents against a database of known signatures. Then came a new breed of offerings that approached the problem in a completely different way - by doing a full protocol analysis on the data stream. Others began to use heuristics or anomaly-based analysis to determine when an attempted attack had taken place.

Today, most IDS/IPS employ a mixture of these detection methods in a single product, though some will be more biased towards one method than another.

According to Cisco, there are five main methods of attack identification (*source: Cisco Systems, The Science of Intrusion Detection System Attack Identification*):

Pattern Matching

Pattern matching in its most basic form is concerned with the identification of a fixed sequence of bytes in a single packet. In addition to the tell-tale byte sequence, most IPS will also match various combinations of the source and destination IP address or network, source and destination port or service, and the protocol. It is also often possible to tune the signature further by specifying a start and end point for inspection within the packet, or a particular combination of TCP flags.

The more specific these parameters can be, the less inspection needs to be carried out against each packet on the wire. However, this approach can make it more difficult for systems to deal with protocols that do not live on well defined ports and, in particular, Trojans, and their associated traffic, which can usually be moved at will.

Although it is often quite simple to define a signature for a particular exploit, basic pattern matching can often be too specific, sometimes requiring multiple signatures to be defined for minor variations in exploits. They are also prone to false positives, since legitimate traffic can often contain the relatively small set of criteria supposedly used to determine when an attack is taking place.

This method is usually limited to inspection of a single packet and, therefore, does not apply well to the stream-based nature of network traffic such as HTTP sessions. This limitation gives rise to easily implemented evasion techniques.

Stateful Pattern Matching

Stateful pattern matching offers a slightly more sophisticated approach, since it takes the context of the established session into account, rather than basing its analysis on a single packet.

Stateful IPS products must consider arrival order of packets in a TCP stream and should handle matching patterns across packet boundaries. Thus, if the exploit string to be matched is *foobar*, and the exploit is split across two packets, with *foo* in one and *bar* in another, the simple packet matching IPS will miss the attack, since it will not be able to match the complete string. The stateful IPS, however, will maintain the session context and reassemble the traffic stream, once again making the complete string available to the detection engine.

This requires more resources than simple pattern matching, since the IPS now has to allocate large amounts of memory and processing power to track a potentially large number of open sessions for as long as possible. This approach does make IPS evasion that much more difficult, though far from impossible.

Direction of traffic is also important here, both in terms of quality of detection and performance.

Client-to-server traffic inspection is the process of applying detection mechanisms to the “request side” portion of a communication - for example, in HTTP this could be the “GET” request coming from a client.

Client-to-server traffic inspection is typically activated to protect all traffic whether internally or externally generated. As the size of the traffic in terms of byte count is relatively small, the processing load placed on the IPS will be lower.

Server-to-client traffic inspection is the process of finding an attack in the "response side" portion of a communication - for example, in HTTP the server-to-client traffic could be the web page and content returned from the server as a result of a "GET" request. Server-to-client traffic, as in this example, is often much larger than the client-to-server traffic in terms of byte count. As a result, the processing load that is placed on an IPS is greater for server-to-client traffic.

Some vendors do not implement server-to-client signatures at all. Often this is for performance reasons, but sometimes it is a design decision by those vendors who also offer HIPS products, which are often better placed to detect the types of exploits executed by malicious response traffic as opposed to request traffic. Some vendors do include server-to-client signatures, but recommend they are disabled when performance is paramount. Bi-directional detection can have a significant impact on performance in some cases - those products which can handle this situation with zero or minimal impact on performance are worth closer inspection (although this level of performance often comes with a higher price tag).

It should be noted that there are situations where disabling server-to-client signatures is reasonably safe, and - happily - these are usually the situations where the highest levels of performance are demanded. Typically, this would be where an IPS is deployed within the network perimeter, where it is unlikely that purely internal HTTP response traffic is likely to be malicious. Perimeter defences would normally be deployed with both client-to-server and server-to-client signatures enabled, but perimeter devices rarely have the same performance requirements as internal ones.

Protocol Decode

Protocol decode IPS take a radically different approach to simple pattern matching IPS products - though sometimes not quite as radically different as the marketing folks would have you believe. With this technique, the IPS detection engine performs a full protocol analysis, decoding and processing the packet contents in the same way that the target client or server application would. It also tends to be stateful.

Although this may seem like using a sledgehammer to crack a nut, it does have the advantage of highlighting anomalies in packet contents much more quickly than doing an exhaustive search of a signature database. It also has the advantage of greater flexibility in capturing attacks that would be very difficult - if not impossible - to catch using pure pattern-matching techniques, as well as new variations of old attacks. These are attacks which - although changing only slightly from variant to variant - would normally require a new signature in the database for the "traditional" IPS architecture, but which would be detected automatically by a complete protocol analysis.

One of the first things the protocol decode engine does is to apply rules defined by the appropriate RFCs to look for violations. This can help to detect certain anomalies such as binary data in an HTTP request, or a suspiciously long piece of data where it should not be - a sign of a possible buffer overflow attempt.

One simple example of how this might work concerns searching Telnet login strings for one of the many well-known login names that rootkits tend to leave behind on the system. A pattern matching system might scan *all* Telnet traffic for *all* these patterns, in which case the more patterns you add, the slower it becomes (not *always* the case, but a reasonable assumption for the purposes of this example).

In contrast, a protocol analysis system will decode the Telnet protocol and extract the login name. It can then perform an efficient search in a binary-search tree or a hash table for just the login name, which should scale much better as new signatures are added.

In theory, therefore, protocol decoding should offer more efficient processing of traffic and improved scalability as more signatures are added, compared to a pure pattern matching solution. In reality, pattern matching solutions rarely opt for a “brute force” approach (there are some extremely intelligent and efficient pattern matching mechanisms available), and so the differences are not always as marked as the marketing people would like us to believe.

Note also, that pattern matching and protocol decoding are not mutually exclusive, as some would lead you to believe. A protocol analysis IPS can only go so far with its protocol decodes before it too will be forced to perform some kind of pattern matching, albeit against a theoretically smaller subset of “signatures”.

One major downside, of course, is that if a completely new type of exploit does surface, it is likely that the developer will have to create new protocol decode code to handle it, whereas the pattern matching approach can allow the administrator to develop a custom signature much more quickly on site.

Protocol decoding does offer a number of advantages, however. It minimises the chance for false positives if the protocol is well defined and enforced (although false positives can be higher if the RFC is ambiguous), and can also be more broad and general to allow the IPS to detect minor variations of an exploit without having to implement separate signatures.

You may see this technique referred to in several different ways:

- *Protocol decode*
- *Protocol Anomaly Detection*
- *Protocol validation*

Each of these terms, if strictly applied, could use a slightly different approach to the problem. For example, we would expect a *protocol decode* engine to perform the sort of additional pattern matching and length checking mentioned above on the field contents in order to detect specific exploits or buffer overflows.

Pure *protocol validation* or *Protocol Anomaly Detection* engines, however, might go no further than decoding just enough to be able to determine if the packet follows the RFC to the letter. If not, they will raise an alert - but in allowing a packet to pass, they cannot be sure that the contents will not contain a means of exploit that just happens to conform with the RFC.

Beware the marketing hype in this particular area – no matter what architecture is used, the performance figures and detection rates in a live deployment will speak for themselves.

Heuristic Analysis

Heuristic-based signatures use some kind of algorithmic logic on which to base their alarm decisions. These algorithms are often statistical evaluations of the type of traffic being presented.

A good example of this type of signature is one that would be used to detect a port sweep. This signature looks for the presence of a threshold number of unique ports being touched on a particular machine. The signature may further restrict itself through the specification of the types of packets that it is interested in (that is, SYN packets). Additionally, there may be a requirement that all the probes must originate from a single source, and even that valid SYN ACK packets must be seen to be returned by the host being probed.

Signatures of this type will react differently on different networks, and can be a significant source of false positives if not tuned correctly, requiring some threshold manipulations to make them conform to the utilisation patterns on the network they are monitoring. This type of signature may be used to look for very complex relationships as well as the simple statistical example given.

Anomaly Analysis

The final approach is to forget about trying to identify the attacks directly, and concentrate instead on ignoring everything that is considered “normal”. This is known as “*anomaly-based*” IPS, and the basic principle is that, having identified what could be considered “normal” traffic on a network, then anything that falls outside those bounds could be considered an “intrusion” - or at the very least, something worthy of note. This is generally better suited to passive IDS rather than in-line IPS devices, given its propensity for false positives.

The primary strength of anomaly detection is its ability to recognise previously unseen attacks, since it is no longer concerned with knowing what an attack looks like - merely with knowing what does not constitute normal traffic. Its drawbacks, of course, include the necessity of training the system to separate noise from natural changes in normal network traffic (the installation of a new - perfectly legitimate - application somewhere on the network, for example).

Changes in standard operations may cause false alarms while intrusive activities that appear to be normal may cause missed detections. It is also difficult for these systems to name types of attacks, and this technology has a long way to go before it could be considered ready for “prime time”.

Which Detection Method Is The Best?

Which detection method to choose is a difficult question, and in all honesty, it is not one with which most of those evaluating these products should concern themselves.

Adequate performance to handle the traffic to which the sensor will be exposed, accuracy of alerts, low incidence of false positives, and centralised management and reporting/analysis tools are far more important than how the packets are processed.

In some instances, the lines blur between methodologies to the point where they become almost indistinguishable.

For example, most protocol decode analysis engines alert the user to the presence of protocol violations that are not directly related to any known attack but are “anomalous” (for example, length-based buffer overflow detection). Therefore, in this instance the engine has attributes of an anomaly-based system.

As we have already mentioned, most protocol analysis systems are also reduced to performing some form of pattern-matching process following the protocol decode. Likewise, even the most basic pattern-matching systems perform some form of protocol analysis - even if it is only for a limited range of protocols. In truth, almost all Network IPS systems are already adopting a hybrid architecture.

By and large, therefore, the *pattern-matching vs. protocol decode* debate is one of religion - something for the marketing departments to shout about. Why should the average user care what happens under the hood as long as the product does what it claims to do - detect and prevent intrusions?

Implementation Challenges

There are a number of challenges to the implementation of an IPS device that do not have to be faced when deploying passive-mode IDS products. These challenges all stem from the fact that the IPS device is designed to work in-line, presenting a potential choke point and single point of failure.

If a passive IDS fails, the worst that can happen is that some attempted attacks may go undetected. If an in-line device fails, however, it can seriously impact the performance of the network.

Perhaps latency rises to unacceptable values, or perhaps the device fails closed, in which case you have a self-inflicted Denial of Service condition on your hands. On the bright side, there will be no attacks getting through! But that is of little consolation if none of your customers can reach your e-commerce site.

Even if the IPS device does not fail altogether, it still has the potential to act as a bottleneck, increasing latency and reducing throughput as it struggles to keep up with up to a Gigabit or more of network traffic. Devices using off-the-shelf hardware will certainly struggle to keep up with a heavily loaded Gigabit network, especially if there is a substantial signature set loaded, and this could be a major concern for both the network administrator - who could see his carefully crafted network response times go through the roof when a poorly designed IPS device is placed in-line - as well as the security administrator, who will have to fight tooth and nail to have the network administrator allow him to place this unknown quantity amongst his high performance routers and switches.

As an integral element of the network fabric, the Network IPS device must perform much like a network switch. It must meet stringent network performance and reliability requirements as a prerequisite to deployment, since very few customers are willing to sacrifice network performance and reliability for security. A NIPS that slows down traffic, stops good traffic, or crashes the network is of little use.

Dropped packets are also an issue, since if even one of those dropped packets is one of those used in the exploit data stream it is possible that the entire exploit could be missed.

Most high-end IPS vendors will get around this problem by using custom hardware, populated with advanced FPGAs and ASICs - indeed, it is necessary to design the product to operate as much as a switch as an intrusion detection and prevention device.

It is very difficult for any security administrator to be able to characterise the traffic on his network with a high degree of accuracy. What is the average bandwidth? What are the peaks? Is the traffic mainly one protocol or a mix? What is the average packet size and level of new connections established every second - both critical parameters that can have detrimental effects on some IDS/IPS engines? If your IPS hardware is operating “on the edge”, all of these are questions that need to be answered as accurately as possible in order to prevent performance degradation.

Another potential problem is the good old *false positive*. The bane of the security administrator’s life (apart from the script kiddie, of course!), the false positive rears its ugly head when an exploit signature is not crafted carefully enough, such that legitimate traffic can cause it to fire accidentally. Whilst merely annoying in a passive IDS device, consuming time and effort on the part of the security administrator, the results can be far more serious and far reaching in an in-line IPS appliance.

Once again, the result is a self-inflicted Denial of Service condition, as the IPS device first drops the “offending” packet, and then potentially blocks the entire data flow from the suspected hacker. If the traffic that triggered the false positive alert was part of a customer order, you can bet that the customer will not wait around for long as his entire session is torn down and all subsequent attempts to reconnect to your e-commerce site (if he decides to bother retrying at all, that is) are blocked by the well-meaning IPS.

Another potential problem with any Gigabit IPS/IDS product is, by its very nature and capabilities, the amount of alert data it is likely to generate. On such a busy network, how many alerts will be generated in one working day? Or even one hour? Even with relatively low alert rates of ten per second, you are talking about 36,000 alerts every hour. That is 864,000 alerts each and every day. The ability to tune the signature set accurately is essential in order to keep the number of alerts to an absolute minimum. Once the alerts have been raised, however, it then becomes essential to be able to process them effectively. Advanced alert handling and forensic analysis capabilities - including detailed exploit information and the ability to examine packet contents and data streams - can make or break a Gigabit IDS/IPS product.

Of course, one point in favour of IPS when compared with IDS is that because it is designed to prevent the attacks rather than just detect and log them, the burden of examining and investigating the alerts - and especially the problem of rectifying damage done by successful exploits - is reduced considerably.

Requirements for effective prevention

Having pointed out the potential pitfalls facing anyone deploying these devices, what features are we looking for that will help us to avoid such problems?

- **In-line operation -** only by operating in-line can an IPS device perform true protection, discarding all suspect packets immediately and blocking the remainder of that flow

- **Reliability and availability -** should an in-line device fail, it has the potential to close a vital network path and thus, once again, cause a DoS condition. An extremely low failure rate is thus very important in order to maximise up-time, and if the worst should happen, the device should provide the option to fail open or support fail-over to another sensor operating in a fail-over group (see below). In addition, to reduce downtime for signature and protocol coverage updates, an IPS must support the ability to receive these updates without requiring a device re-boot. When operating inline, sensors rebooting across the enterprise effectively translate into network downtime for the duration of the reboot
- **Resilience -** as mentioned above, the very minimum that an IPS device should offer in the way of High Availability is to fail open in the case of system failure or power loss (some environments may prefer this default condition to be “fail closed” as with a typical firewall, however - the most flexible products will allow this to be user-configurable). Active-Active stateful fail-over with cooperating in-line sensors in a fail-over group will ensure that the IPS device does not become a single point of failure in a critical network deployment
- **Low latency -** when a device is placed in-line, it is essential that its impact on overall network performance is minimal. Packets should be processed quickly enough such that the overall latency of the device is as close as possible to that offered by a layer 2/3 device such as a switch, and no more than a typical layer 4 device such as a firewall or load-balancer.
- **High performance -** packet processing rates must be at the rated speed of the device under real-life traffic conditions, and the device must meet the stated performance with all signatures enabled. Headroom should be built into the performance capabilities to enable the device to handle any increases in size of signature packs that may occur over the next three years. Ideally, the detection engine should be designed in such a way that the number “signatures” (or “checks”) loaded does not affect the overall performance of the device.
- **Unquestionable detection accuracy -** it is imperative that the quality of the signatures is beyond question, since false positives can lead to a Denial of Service condition. The user MUST be able to trust that the IDS is blocking only the user selected malicious traffic. New signatures should be made available on a regular basis, and applying them should be quick (applied to all sensors in one operation via a central console) and seamless (no sensor reboot required)
- **Fine-grained granularity and control -** fine grained granularity is required in terms of deciding exactly which malicious traffic is blocked. The ability to specify traffic to be blocked by attack, by policy, or right down to individual host level is vital. In addition, it may be necessary to only alert on suspicious traffic for further analysis and investigation
- **Advanced alert handling and forensic analysis capabilities -** once the alerts have been raised at the sensor and passed to a central console, someone has to examine them, correlate them where necessary, investigate them, and eventually decide on an action. The capabilities offered by the console in terms of alert viewing (real time and historic) and reporting are key in determining the effectiveness of the IPS product.

The NSS Intrusion Prevention Group Test

The NSS Group conducted the first comprehensive IPS test of its kind, now updated in this Edition.

This exhaustive review will give readers a complete perspective of the capabilities, maturity and suitability of the products tested for their particular needs.

As part of its extensive IPS/Attack Mitigator test methodologies (see section on *Testing Methodology* later in this report for detailed methodologies, updated for this latest test) The NSS Group subjects each product to a brutal battery of tests that verify the stability and performance of each IPS tested, determine the accuracy of its security coverage, and ensure that the device will not block legitimate traffic.

If a particular IPS has been designated as *NSS Approved*, customers can be confident that the device will not significantly impact network/host performance, cause network/host crashes, or otherwise block legitimate traffic.

To assess the complex matrix of IPS/Attack Mitigator performance and security requirements, the NSS Group has developed a specialised lab environment that is able to exercise every facet of an IPS product. The test suite contains over 800 individual tests that evaluate IPS products in three main areas: *performance and reliability, security accuracy,* and *usability*.

This thorough review should give readers a complete perspective of the capabilities, maturity and suitability of the products tested for their particular needs.

Performance

Any IPS is expected to be reliable (not crash), to never block legitimate traffic, and to not unduly affect network or host system performance.

The latency and throughput of a Network IPS (NIPS) or Attack Mitigation device must be on a par with other equipment in the network on which it is deployed, and in this respect, an in-line NIPS must strive to perform much more like a switch than a typical passive security device, especially when it is necessary to install more than one NIPS in the same data path.

Detection/Blocking Performance Under Load

This group of tests verifies that the IPS does not adversely impact legitimate traffic, even when new TCP connections are being created rapidly. We also verify that the sensor is capable of detecting and blocking exploits when subjected to increasing loads of background traffic up to the maximum bandwidth supported as claimed by the vendor. An IPS that misses attacks under load can be evaded. An IPS that adversely affects legitimate background traffic will not stay in-line for long.

A fixed number of exploits are launched with zero background traffic to ensure the sensor is capable of detecting our baseline attacks. Once that has been established, increasing levels of varying types of background traffic are generated **through** the IPS device in order to determine the point at which the sensor begins to miss attacks.

All tests are repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic (or up to the maximum rated throughput of the device in 25 per cent increments should this be less than 1Gbps). The test is conducted with UDP, HTTP, and mixed-protocol traffic and includes packet rates up to 453,000 packets per second and connection rates up to 20,000 connections per second.

Latency & User Response Times

In any network environment latency is important. Latency may impose an upper bound on throughput and it also has an impact on interactive applications, thus affecting user response time. As such, it is important to understand the impact of latency introduced by a NIPS and to determine the maximum acceptable delay, which will be different for each network.

There is a direct relationship between latency introduced by a networking device and the maximum throughput allowed by that device on a single TCP connection. There is a critical value for the *round trip time* (RTT) of a packet in each network, and if the latency is below this critical value, TCP throughput will be unaffected - instead, it is the line speed of the underlying network which becomes the bottleneck. Above this critical value, however, TCP throughput is negatively impacted. To be specific, the maximum throughput achievable for any given TCP connection in a zero loss network is expressed as:

throughput = window / RTT

where *window* is the maximum TCP window size (64 Kbytes by default) and RTT is the round trip time in the network.

This equation tells us that the throughput of a TCP connection is inversely proportional to network latency (note that this is TCP throughput for *one* connection - the aggregate bandwidth is not affected by latency). In other words, if you double latency, you halve throughput.

Consider adding a NIPS in an internal Gigabit network where the RTT is 200 microseconds. The critical value for RTT in a Gigabit network is 500 microseconds (below which it may no longer be possible to achieve 1Gbps of throughput), which means the NIPS can add a maximum of 300 microseconds to the RTT without affecting the network. In this particular case, therefore, for an internal, high speed deployment, the administrator may determine that his chosen IPS device needs to be capable of sub-300 microsecond latency under normal traffic loads.

Of course, the latency of an IPS device may vary significantly based on packet size, complexity of the protocol, presence of attack traffic, or simply the makeup of the normal traffic passing through it. For example, Gigabit segments, will rarely carry only a single TCP connection. Rather, a saturated Gigabit segment could be supporting hundreds, if not thousands of TCP connections, and this multiplexing eases the impact of latency on the overall throughput on the segment.

Although each of these connections carries only a fraction of the total throughput, a few connections tend to dominate. The maximum latency for a NIPS is then determined by the utilisation of the fastest connection. For example, in a Gigabit Ethernet segment carrying 10,000 TCP connections the fastest connection might have a throughput of 250Mbps. In this case, the critical value for round trip latency is as high as 2 milliseconds.

Assuming the latency without the NIPS is 300 microseconds, an administrator may therefore determine that his chosen NIPS device must be capable of 1700 microsecond round trip latency (850 microseconds in each direction).

Such critical value calculations are important when TCP connections achieve maximum throughput, which is true for large data transfers.

For smaller data transfers, and non-TCP applications like NFS, latency has a more direct impact on user experience - response time is directly proportional to latency. That is, *doubling latency doubles response time.* In these situations, the latency of the network in which a NIPS is deployed determines the acceptable latency of the NIPS.

Consider deploying a hypothetical NIPS with 1 millisecond one-way latency in the following scenarios:

- In internal corporate LANs, the round trip latency could be in the 200-300 microsecond range. Deploying our hypothetical NIPS would increase the maximum round trip latency to 2.3 milliseconds, an increase of just over 700 per cent. The time to copy a large group of files, for example, would increase by a factor of seven.
- In inter-campus corporate networks connected over a MAN, the latency could be in the 500-1000 microsecond range (or less). Deploying our hypothetical NIPS would increase the maximum round trip latency to 3 milliseconds, a minimum increase of 300 per cent. The time to copy a large group of files, for example, would increase by at least factor of three.
- Internet facing connections experience round-trip latency from 10-100 milliseconds. Deploying our hypothetical NIPS would increase the round trip latency by 1-10 per cent, which would have only a minor impact on the user experience.

The latency of the NIPS must therefore be evaluated in the context of the network in which it is deployed. For example, to protect networks that are accessed over the public Internet, one-way NIPS latencies in the 1-2 millisecond range would be acceptable. Whereas for NIPS deployments on MAN/WAN links, NIPS latencies of well under 1 millisecond would be essential. And as we have already mentioned, for deployments on internal networks where latencies are a few hundred microseconds, NIPS latencies of less than 300 microseconds would be more appropriate.

Network administrators have laboured long and hard to reduce latency within the corporate network to an absolute minimum. Core network devices such as switches are frequently chosen as much on their performance - packet loss and latency under all load conditions - as any other feature. Given that Network IPS devices are operating in-line, it is not surprising that they will be evaluated in a similar way.

For this reason, part of The NSS Group methodology uses very similar testing techniques to those we would normally employ when testing switches (in order to determine *packet latency*), in **addition** to measuring *application latency*. This group of tests determine the effect the IPS sensor has on the traffic passing through it under various load conditions. High packet latency will lower TCP throughput. High application latency will create a negative user experience.

Bi-directional network latency of a range of differently-sized UDP packets is measured under three test conditions: with no load, with 500 Mbps of HTTP traffic (or half the rated load of the device if this is less than 1Gbps), and while the device is under a heavy SYN flood attack (up to 10 per cent of the rated throughput of the sensor).

Spirent Avalanche and Reflector devices are also used to generate HTTP sessions through the device in order to gauge how any increases in latency will impact the user experience in terms of failed connections and increased Web response times.

This "*application latency*" is measured both with no background load and while the device is under attack.

Stability & Reliability

These tests verify the stability of the IPS device under various extreme conditions. Long-term stability is critical for an in-line IPS device, where failure can produce network outages.

In the first part of this test, we expose the external interface of the sensor to a constant stream of attacks over an extended period of time. The device is configured to block and alert, and thus this test provides an indication of the effectiveness of both the blocking and alert handling mechanisms. A continuous stream of exploits mixed with some legitimate sessions is transmitted through the sensor at a maximum rate of 90 per cent of the claimed throughput of the device for eight hours with no additional background traffic.

The device is expected to remain operational and stable throughout this test, blocking 100 per cent of recognisable exploits, raising an alert for each, and passing 100 per cent of legitimate traffic. If any recognisable exploits are passed - caused by either the volume of traffic or the IPS device failing open for any reason - this will result in a FAIL. If any legitimate traffic is blocked - caused by either the volume of traffic or the IPS device failing closed for any reason - this will also result in a FAIL.

In the second part of the test we stress the protocol stack of the device under test by exposing it to malformed traffic from the ISIC test tool for eight hours. The device is expected to remain operational and capable of detecting and blocking exploits throughout the test to attain a PASS.

We scan the management interface for open ports and active services and report on known vulnerabilities. We also stress the protocol stack of the management interface of the NIPS by exposing it to malformed traffic from the ISIC test tool. The device is expected to remain (a) operational and capable of detecting and blocking exploits, and (b) capable of communicating in both directions with the management server/console throughout the test to attain a PASS. We also note whether the sensor detects the ISIC attacks even though targeted at the management port.

Security Effectiveness

Detection Accuracy & Breadth

This group of tests verifies that the NIPS will not block legitimate traffic (*Accuracy*) and is capable of detecting and blocking a wide range of common exploits (*Breadth*). Although *breadth* is extremely important, *accuracy* is critical because a NIPS that blocks legitimate traffic will not remain in-line for long.

We have a number of trace files of normal traffic with "suspicious" content, together with several "neutered" exploits that have been rendered completely ineffective. The IPS attains a "PASS" for each test case if it does **not** raise an alert and does **not** block the traffic. Whilst it is not possible to validate completely the entire signature set of any IPS, this test demonstrates how accurately the IPS detects and blocks a wide range of common exploits, port scans, and Denial of Service attempts.

This test is repeated twice: the first run with blocking disabled on the IPS in order to determine which attacks are detected and how accurately they are detected (*Attack Recognition Rating*); the second run with blocking enabled in order to determine which attacks are blocked successfully regardless of how they are detected or what alerts are raised (*Attack Blocking Rating*).

Following the initial test run, each vendor is provided with a list of CVE references of the attacks missed and is allowed 48 hours to produce an updated signature set. This updated signature set must be released to the general public as a standard signature/product update before the report is published - this ensures that vendors do not attempt to code signatures just for this test.

Naturally, Rate-Based IPS devices will not respond to the same attack traffic as Content-Based devices, and so for those the Detection Accuracy tests involve detecting and mitigating a wide range of rate-based attacks such as port scans, SYN floods, connection floods, and so on. We note which of these are mitigated completely, which are mitigated partially, and which require the use of built-in firewall capabilities.

Resistance To Evasion Techniques

These tests verify that the IPS is capable of detecting and blocking basic exploits when subjected to varying common evasion techniques. An IPS that cannot detect attacks subjected to these “script kiddie” evasion techniques is easily bypassed.

The tests consist of four parts (only the third is applicable to Rate-Based devices):

- ***Baselines -*** *This establishes that the IPS is capable of detecting and blocking a number of common basic attacks (our baseline suite) in their normal state, with no evasion techniques applied.*
- ***Packet Fragmentation and Stream Segmentation -*** *The baseline HTTP attacks are repeated, running them through fragroute using 19 evasion techniques.*
- ***URL Obfuscation -*** *The baseline HTTP attacks are repeated, this time applying 9 URL obfuscation techniques made popular by the Whisker Web server vulnerability scanner.*
- ***Miscellaneous Evasion Techniques -*** *Certain baseline attacks are repeated, and are subjected to 7 protocol- or exploit-specific evasion techniques, including altering default ports, inserting spaces in FTP command lines, inserting non-text Telnet opcodes in FTP data streams, and RPC record fragging.*

For each of the evasion techniques, we note if (i) the attempted attack is blocked successfully (the primary aim of any IPS device), (ii) the attempted attack is detected and an alert raised in **any** form, and (iii) if the exploit is successfully “decoded” to provide an accurate alert relating to the original exploit, rather than alerting purely on anomalous traffic detected as a result of the evasion technique itself.

Stateful Operation

If the IPS is tracking TCP session state, then it has the potential to introduce denial of service when the session table becomes full (too many connections) or if it can’t keep up with the creation of new sessions (too many connections per second).

As with latency and bandwidth, the number of connections supported by the IPS and its connection per second rate should be matched to the network.

For example, a fully saturated Gigabit Ethernet link can handle 22,000 5KByte transfers per second. Assuming each connection lasts 20 seconds, the IPS should be able to handle 448,000 simultaneous connections. These numbers scale proportionately for slower networks. Any IPS that doesn't offer these capabilities will impact performance of Web or e-commerce servers.

The aim of this section is to be able to determine whether the IPS is capable of monitoring stateful sessions established through the device at various traffic loads without either losing state or incorrectly inferring state.

An IPS that does not maintain TCP session state can flood the management console with false-positive alerts. Although this should not directly impact the IPS blocking function, it can make it very hard to perform forensic analysis of the attacks. In addition, if the default condition of the sensor is to block all traffic for which it does not believe there is a current connection in place, then an inability to maintain state under extreme conditions could result in the sensor blocking legitimate traffic by mistake.

In the first part of this test, we transmit a number of packets taken from capture files of valid exploits, but without first establishing a valid session with the target server. In order to receive a "PASS" in this test, no alerts should be raised for any of the actual exploits. However, each packet should be blocked if possible since it represents a "broken" or "incomplete" session.

In part two, we test whether the sensor is capable of preserving state across increasing numbers of open connections, as well as continuing to detect and block new exploits while not blocking legitimate traffic when the state tables are filled. Various numbers of TCP sessions from 10,000 to 1,000,000 (one million) are tested.

This test is run in both the out-of-box configuration and then repeated after applying any tuning recommended by the vendor (if applicable) to increase the size of the state tables.

Usability

After quantitatively evaluating the network performance and security effectiveness of the IPS, we qualitatively evaluate the features and usability of the product.

This evaluation provides the reader with valuable insight into product features, how easy it is to install the IPS and perform common, day-to-day operations with the management console. Areas evaluated include *installation, configuration, policy editing, alert handling,* and *reporting and analysis*.

NFR SENTIVIST SMART SENSOR 100C V1.3

Executive Summary

NFR offers a range of Sentivist appliances for different deployment scenarios, and the IPS sensors can be deployed in in-line passive (learning mode, with no blocking), passive IDS, in-line IPS (fail closed) and in-line IPS (fail open) modes.

The Smart Sensor 100C tested is a 1U rack mount appliance with four copper Gigabit ports which can be used to monitor three segments (3 ports) in passive IDS mode, or a single segment (2 ports) in in-line IPS mode. NFR rates this device at 100Mbps

Blocking rates were good out of the box, and improved to 92 per cent following an update, and resistance to most evasion techniques was excellent. Throughput and latency were excellent under all network loads and across all packet sizes up to the rated 100Mbps, and under normal network conditions we would expect the device to handle beyond this level of traffic.

We also found the Smart Sensor 100C to be very stable and reliable under extended attack, although resistance to high-levels of certain types of DOS/DDOS attacks was poor. We would recommend deploying this device behind an additional layer of protection, such as an attack mitigator or firewall with DOS/DDOS mitigation capabilities.

Management and configuration of multiple Sentivist devices is made simple via the scalable, three-tiered architecture of the Sentivist Enterprise Console. Although policy definition can be daunting for those new to the idea of NFR's Backends and Packages, the system is actually extremely flexible and powerful. The use of Policy Groups and sub-Groups and the concept of inheritance from higher-level groups down to lower-levels, and even down to individual Sensors, makes the task of deploying policies across large organisations very straightforward.

Alert handling and reporting are very comprehensive, typical of a company with such a strong IDS pedigree. Multiple Alert Browser windows can be created, each one with customised column layouts and data filters, and these can be saved for subsequent re-use, each restricted to a different administrator if required. Accessing alert data can be performed in a variety of ways, and the unique Timeline view and useful event correlation capabilities make it easy for administrators to focus on the most important events.

Overall, both policy management and alert handling capabilities are amongst the best we have seen in our labs.

Architecture

NFR Sentivist systems are based on a three-tier architecture only - there is no direct Sensor management capability in the current version. Although this clearly provides reliability and scalability, it does mean that a separate, dedicated host is always required for the management server component, possibly making it a less attractive (and more expensive) proposition for those who require only one or two sensors.

Each Sentivist system includes the following components:

- *Sentivist Enterprise Console (includes Enterprise Server and Enterprise Console components)*
- *Sentivist Server*
- *Sentivist Smart Sensor*

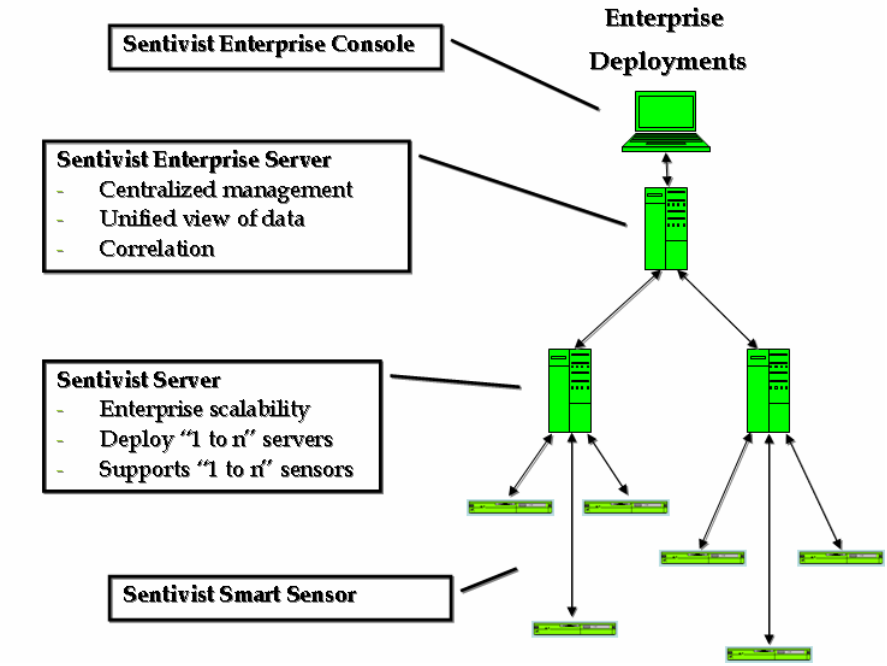

Figure 1 - Sentivist: Architecture

Sentivist Server

Sentivist Server receives, processes, and manages alert and event data generated by Sentivist Sensors. It also provides a centralised facility for Sentivist Sensor management and for viewing captured data.

Key functions of Sentivist Server are:

- *Short-term storage of alert and event data*
- *Centralized management of Sentivist Sensor configuration data*
- *Support for alert and event data queries using Sentivist Enterprise Console*
- *Integration with external applications for alert notification*
- *Audit of actions and events affecting the Sentivist system state and configuration*

All communications between the Sentivist Server, Sensors, Administration Interface and Enterprise Console are performed over AES encrypted channels.

Sentivist Enterprise Console

Enterprise Console (EC) is the management component of the Sentivist System. Through Enterprise Console, the administrator can monitor alerts generated throughout the entire system and perform high-level analysis.

Enterprise Console also allows configuration and management of all Sentivist Servers and Sentivist Sensors deployed across the corporate network from a central point.

Enterprise Console has two components:

- ***Sentivist Enterprise Server*** *is a centralised server that aggregates information from multiple Sentivist Servers*
- ***Sentivist Enterprise Client*** *is a Java-based user interface providing interaction between the administrator and the Enterprise Server*

Sentivist EC can be installed on the same host as the Sentivist Server component if required, and this is typical in most small deployments. In larger deployments, however, it is possible to improve scalability by installing EC on a dedicated server communicating with one or more Sentivist Servers.

Administration Interface (AI)

The *Administration Interface* (AI) is the original Windows-based management application, which can be installed on any Windows 2000/XP host on the network. This is being superseded by the EC, but is still distributed as part of the current system.

The NFR AI provides administrator access for querying data, viewing alerts, and configuring the Sensor. Multiple Sensors can be managed from a single AI, which communicates directly with the Sentivist Server over an encrypted channel.

Sentivist Smart Sensor

Each Sentivist system will have one or more Sentivist Sensors, each of which is capable of operating in passive IDS, in-line passive (learning only, no blocking) or in-line IPS (full blocking, fail-open or fail-closed) modes. Configurable response mechanisms determine what action to take when Sentivist Sensor detects an intrusion, including alert generation, data recording, resetting of the TCP session, and changing of firewall rules.

Sentivist Sensor is normally delivered as a pre-configured appliance consisting of the appropriate hardware and software combination for each Sensor model. A software-only version is also available.

A range of models are available for different deployment scenarios - the model provided for testing was the 100Mbps Sentivist Smart Sensor 100C. This is a 1U rack mount appliance based on standard Intel hardware with a single Xeon processor and 2GB RAM. A large hard disk is used to store the operating system and IPS code (this is no longer stored on CD-ROM during operation as with previous models) and also for local logging of alerts.

The device sports a total of four copper Gigabit ports, one of which is dedicated to management. The remaining three ports can be used to monitor a single segment in in-line mode, or three segments in passive IDS mode. It is not possible to configure a mixed mode, with two ports operating in-line and one in passive IDS.

The ports provide a hardware bypass capability allowing the Smart Sensor 100C to operate in a fail-closed or fail-open mode when in-line.

One feature which is not common amongst the competition is the ability to specify a user-defined maximum latency figure, beyond which the device will cease inspection. Although the device will still not pass packets in fail-closed mode, in conjunction with the fail-open mode of operation this allows the Smart Sensor 100C to temporarily “step out of the way” when overwhelmed by network traffic, thus preventing it from becoming a point of failure in the network (although obviously at the expense of security protection, at the administrator’s choice).

There are no additional fault tolerant/High Availability options available at the Sensor level in the current release (although Sensors can fail over to a second Sentivist Server should the first fail, providing resilience at the management level).

Sentivist Sensors monitor a particular network for suspicious activity and attacks. These incidents are detected by the ***Packages*** and ***Backends*** installed and enabled on each Sensor. A Package targets a particular area of concern (for example, Web traffic). Backends are variations within a Package - they target specific types of incidents (for example, attacks against Apache Web servers, attacks against ColdFusion servers, and so on). Some Backends also have ***Variables*** that can be customised (for example, the time period used to buffer ColdFusion attacks).

When a Sensor detects a possible incident, it generates an ***alert signature***, which includes the name of the Package and Backend (for example, *www_coldfusion:buffered_alert*) and an alert ID number.

At the Sentivist Server level, the alert signature is mapped to an ***alert type*** (for example, “*Buffered ColdFusion alerts*”). The alert type has a corresponding name, priority, and description (known collectively as the ***alert definition***). The alert type also has one or more corresponding ***alert rules***, which tell the Server what action or actions to take.

Typically, the alert type has a rule that records the alert to the ***networklist***, one of three default locations in the database (the others are the ***systemlist***, for alerts related to routine system activity, and the ***auditlist***, for alerts related to internal security). Additional alert rules can be used to generate e-mails, display the alert in a pop-up window, send the alert to an external program, and so forth.

For ease of administration, similar alert types are organised into ***alert groups***. Through alert groups, alert rules can be associated with multiple alert types at one time.

Alerts that are recorded to the *networklist* can be viewed through an alert browser window. Alerts are viewed through the Sentivist Administration Interface at the Sentivist Server level, or the Enterprise Console at the enterprise level.

The Enterprise Server receives alerts from all Sentivist Servers in the system. At this level, a different type of rule (known as a ***correlator***) goes into effect. A correlator tells the Enterprise Server to take action if it detects alerts that fit a specified pattern (for example, 1000 alerts with the same Source IP within two minutes).

Two types of correlators are available. ***Boolean correlators*** detect alerts that contain specified values (for example, alerts with the Source IP of 10.34.56.8).

Cluster correlators detect alerts with the same value in specified fields, regardless of the actual value (for example, alerts with the same destination port). Cluster correlators can be used in conjunction with alert groups (for example, alerts related to Web traffic and protocols).

Another dimension that appears at the enterprise level is the ***site***. The site field is typically used to identify Sentivist Servers that are in the same physical location (such as a city and state). It is possible, therefore, to detect alerts that were generated from the same physical location. In addition, ***site filters*** can be used to identify and ignore benign traffic from a particular site.

All system settings can be configured at the Enterprise level, including Packages and Backends, Variables, alert signature mappings, alert definitions, alert rules, and alert groups. For ease of management, Package, Backend, and Variable settings can be copied from one machine to another. These settings are known collectively as a ***policy***. Furthermore, system components are organised into ***Policy Groups***. Through these Groups, configuration settings can be applied quickly and uniformly throughout the entire system.

Backends

A Backend is one of the basic components of a Sentivist Sensor, controlling the type of data that is written and how it is recorded. For example, different Backends included gather information about TCP and UDP traffic. Each Backend consists of several components: a *Filter*, *configuration files*, and a *Recorder*.

- ***Filters -*** *The Filters are written in N-Code, NFR's proprietary event-driven language (N-Code training is available), and list the events that cause the Smart Sensor engine to begin gathering data. For example, the N-Code for the UDP Backend begins recording data from packets if the packet contains a UDP header. The Filter also tells the engine what to do with the data, and functions available in N-Code tell the engine what kind of data to record and then pass that data on to the Recorder. For example, the Pingflood Detector Backend tells the Smart Sensor engine to track source and destination IP addresses, the number of packets sent, and the elapsed time since the last packet. All of this information is sent to the Recorder. Other functions in N-Code tell the engine what data should be sent as an alert. For example, the Pingflood Detector backend includes N-Code that tells the Smart Sensor engine to send an alert to the alert manager when more than a certain number of ping packets are sent to a certain IP address in a certain time.*
- ***Configuration files -*** *The configuration files provide information about the title of the Backend and other information displayed via the EC. More importantly, the configuration files detail the types of data that are being examined. For example, the configuration files for the UDP backend indicate that the first column of data recorded is a source port and should be labelled as "Source Port" on the EC display.*
- ***Recorders -*** *The Recorder writes the information gathered by the Backends to files.*

Note that because security events are defined using N-Code routines, it is possible for end users and administrators to modify the built in "signatures" where required.

New signatures can also be written from scratch, and the entire NFR signature library is open to inspection and modification. This makes NFR Sentivist one of the most extensible and flexible IDS/IPS systems on the market.

N-Code is compiled on the sensor as packages are deployed from the Sentivist Server in order to make it more efficient.

Packages

Backends are stored in logical groups called *Packages*. For example, the *Denial of Service Detection Package* included with the Sensor includes a variety of Backends that watch for various methods of denying network services. Backends can also share N-Code, which is helpful when there are several Backends that need to process the same data, thus helping to eliminate redundant processing.

A large library of signatures is available, and is being enhanced constantly by NFR's *Rapid Response Team* (RRT). New signatures are placed on a secure NFR support site as soon as they are available and customers are emailed with details of the attack they address and other pertinent information. A secure download process enables administrators to quickly acquire and distribute the new signatures to all Sensors in the organisation via the Sentivist Server and EC. One point worth noting here is that NFR does not wrap all its Backends and Packages in to a single monolithic signature set, and there is no facility to download and install all new and updated signatures in a single operation. Instead, it is up to the administrator to select, download and install each individual Package.

Although this can often be a laborious process, NFR maintains that it prefers to allow the administrator to make the final decision on exactly which packages should be downloaded and installed on his system (i.e. why bother downloading the latest batch of IIS signatures if you only use Apache?) - certainly a valid approach, but somewhat contrary to the current "*click and forget*" philosophy adopted by some vendors in the IPS space.

Performance

The aim of this section is to verify that the sensor is capable of detecting and blocking exploits when subjected to increasing loads of background traffic up to the maximum bandwidth supported as claimed by the vendor.

For each type of background traffic, we also determine the maximum load the IPS can sustain before it begins to drop packets/miss alerts. It is worth noting that devices which demonstrate 100 per cent blocking but less than 100 per cent detection in these tests will be prone to blocking **legitimate** traffic under similar loads.

The NFR Sentivist Smart Sensor 100C was tested up to 100Mbps, the rated speed of the device, and performance at all levels of our load tests was impeccable, with 100 per cent of all attacks being detected and blocked under all load conditions. We would happily confirm NFR's 100Mbps rating for this device, and would deem it conservative in most normal networks.

The basic latency figures of the NFR Smart Sensor 100C were excellent for a 100Mbps device across the board under all traffic loads.

They ranged from 136µs with 25Mbps of 256 byte packets, to 198µs with 100Mbps of 1000 byte packets.

Behaviour throughout the tests with no background traffic was consistent and predictable, with small increases as additional network load was applied from 25Mbps to 100Mbps. Placing the device under a half load of 50Mbps of HTTP traffic increased latency slightly, rising from 136µs to 160µs with 256 byte packets, from 155µs to 217µs with 550 byte packets, and from 195µs to 217µs with 1000 byte packets.

Given such low latencies, HTTP response times were unusually high with an average of 246ms with 50Mbps of HTTP traffic. The device also had significant problems with our DDOS test, and would perform best when situated behind an attack mitigation device or a firewall with DDOS mitigation capabilities. Latency when under 10Mbps (14800 packets per second) of SYN flood traffic was excessive, although UDP packet loss was minimal. The overall effect on HTTP traffic, however, was an increase in response times and a high number of failed transactions.

This is an issue which NFR recognises and is working on at the time of writing. It may be possible to configure the Smart Sensor to perform better in these situations by disabling all notice session-based filters (there were four of these enabled during the test) though we did not verify this and are not sure of the resulting effect on overall protection as a result.

Apart from the DDOS issue, the Smart Sensor 100C performed consistently and completely reliably throughout our tests. Under eight hours of extended attack (comprising millions of exploits mixed with genuine traffic) it continued to block 100 per cent of attack traffic, whilst passing 100 per cent of legitimate traffic.

Exposing the sensor interface to ISIC-generated traffic had no long-term adverse effect on the device, although communications between the management console and the management server/sensor were intermittent during the attack.

Please refer to the *Testing Methodology* section for full details of the methodology used and complete performance results.

Security Effectiveness

We installed one sensor with the latest updates, and enabled all attack/exploit signatures (recorders, audit and policy enforcement signatures were disabled, along with four packages which NFR recommends to be disabled in high performance environments).

Out of the box, signature recognition was good at 83 per cent, and was improved to 92 per cent following the application of a signature update after 48 hours. Blocking performance was identical following the update.

Performance in our “false negative” tests was adequate out of the box, and improved slightly following the signature update. However, there were still three misses out of the 14 test cases following the signature update, which was unusual given that NFR has a good track record for writing signatures for vulnerabilities rather than exploits. The product also did very well at detecting most of the variants that we include alongside our basic test cases.

A major concern in deploying an IPS is the blocking of legitimate traffic. The device failed two test cases in our false positive test suite, and this was reduced to one (fake FireWall-1 RDP traffic) following the signature update. No other false positives were noted during stress testing with either Avalanche traffic or real-world traffic mixes, however.

Resistance to known evasion techniques was excellent, with the Smart Sensor 100C achieving a clean sweep across the board in most of our evasion tests. *Fragroute* and *Whisker* both failed to deceive the device into ignoring valid attacks, and all of the attempts were decoded accurately. Of the miscellaneous evasion techniques, changing ports on backdoors, inserting spaces into FTP commands and non-text Telnet opcodes in FTP data streams all proved troublesome, but the rest (including extensive RPC record fragging) were handled well.

NFR does not provide definitive open connection figures due to the workings of the "*pressure system*" architecture (see *Testing Methodology* section for further details) which will see the maximum figure vary depending on current network load. During testing, we verified the Smart Sensor 100C's ability to handle up to 140,000 simultaneously open connections whilst successfully maintaining state on our half-open exploits.

During testing, the default operation of the device was to reject new connections when the state tables were full or resources were low, and this meant that once we exceeded the connection limit the device continued to maintain state on existing connections (thus detecting our half-open exploits), but inevitably, as a consequence, began to block legitimate traffic. This behaviour is not configurable.

Unusually, stateless "exploits" are alerted upon by default, and blocked. We do not believe that this is correct behaviour, since it leaves the device vulnerable to *Stick* and *Snot* tools (although the traffic would be blocked, of course). It is possible to configure the device to suppress alerts and pass the mid-flow traffic, but our preferred configuration - suppress alerts and block mid-flows - is not possible.

Please refer to the *Testing Methodology* section for full details of the methodology used and complete performance results.

Usability

This part of the test procedure consists of a subjective evaluation of the features and capabilities of the product, and covers *installation, configuration, policy editing, alert handling,* and *reporting and analysis*.

Installation

Installation of the complete Sentivist system is in several stages, given the number of components to install, but is straightforward. Individual Sensor deployment is very quick and easy. The Sensor is normally provided as an appliance, and a command-line configuration script provides the means to enter initial parameters such as IP address, address of Sentivist Server which will manage the Sensor, pre-shared key for Server communication, time zone, date and time, NTP details, operating mode (passive IDS, inline fail-severed, inline fail-passthrough or inline bridge (where packets are inspected and alerts raised, but no packets are dropped)), and which interfaces are used for monitoring and management.

These settings can all be saved to a floppy disk (or created manually in a text editor on floppy) to be used to reinstall the sensor at a later date in "unattended" mode. This also provides the means for administrators to create configuration floppies to be shipped to remote sites for "lights out" installations.

Once the Sensor is operational, a real-time monitoring screen is displayed on the console which shows CPU utilisation, packet capture statistics, memory usage, disk usage, and total number of Packages and Backends installed. We would like to see this extended to show current open connections, details of the most recent alerts to be logged locally, and number of alerts currently queued for transmission to the Sentivist Server, all of which could be invaluable in troubleshooting situations.

A password-protected screen on the console provides access to a text-based menu allowing modification of all the Sensor parameters set during installation, plus the ability to restart or halt the Sensor.

The Sentivist Server and Enterprise Console software are both installed via straightforward scripts from CD onto any RedHat (V9 or Enterprise V3.0) or Solaris (V8 or V9) server which meets the recommended system requirements. The underlying database software is installed automatically if required - MySQL is used for the Server and PostGreSQL for the EC, although the EC will be moved to MySQL also in the near future (an Oracle option is also available).

The Enterprise Client is a pure Java-based application which can be installed on any Windows (2000/XP), RedHat (V9 or Enterprise V3.0) or Solaris (V8 or V9) host.

Documentation is provided as both hard and soft copy versions, and consists of a *Getting Started Guide*, *Sentivist User Guide* and *Enterprise Console User Guide*. All of the documentation is well written and extremely thorough.

Configuration

On firing up the *Enterprise Console (EC) Client*, the administrator is presented initially with the *Alert Browser* window. All of the main management and configuration functions, however, are available from the *Management* menu option (there is a toolbar and menu bar along the top of the screen).

A user name and password is required to log in to the EC, and policy settings control what a particular user can see and do within the Console after successfully logging in. The Enterprise Console displays only the functions and devices that each individual administrator is authorised to use based on the user credentials supplied.

Each new user can be created as a *Normal* user or *Administrator*. *Normal* users can view alerts and create views, and if a normal user has permission to restart one or more Sentivist Sensors, that user will also be able to view (but not modify) policy groups. *Administrators* have complete access to all functions. Check boxes are used to specify if a user is allowed to restart Sensors or interrogate audit information for any Sensor to which he has access.

A hierarchical tree display at the bottom of the screen lists all Sites, Servers and Sensors, and check boxes at each level can be used to determine those Sensors to which the user will have access. By selecting a Server, the user is given access to all Sensors reporting to that Server (and any Sensors which are added to that Server in the future). Otherwise, access can be provided to individual Sensors as required.

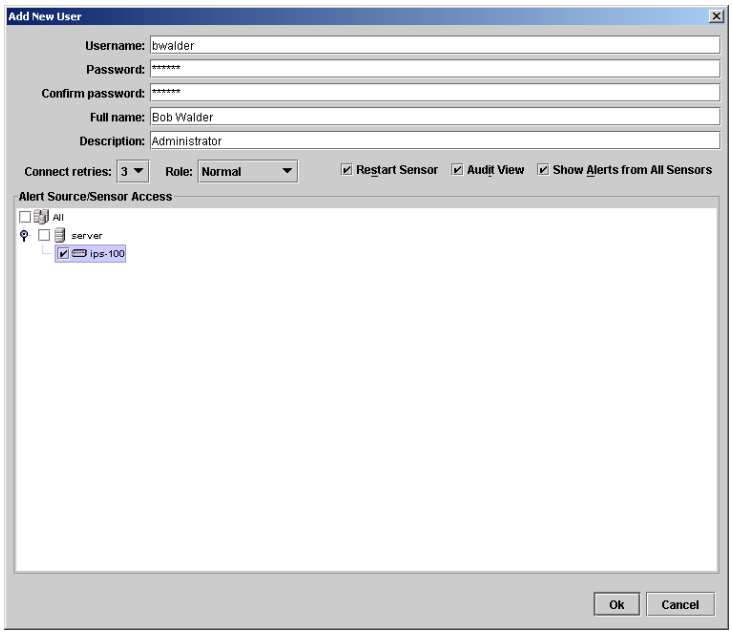

Figure 2 - Sentivist: Creating user accounts

Enterprise Console has more than 100 built-in alert types related specifically to the security of the Sentivist system. Security-related events are recorded to a special *auditlist*, which can be displayed if the user has the *audit view* permission enabled. These events include unauthorised attempts to:

- *Log in to Sentivist Servers*
- *Modify alert types and alert rules*
- *Modify Packages, Backends, and Variables*
- *Retrieve or view alert or event data*
- *Modify user permissions*
- *View the auditlist or modify audit alert types and rules*

At least one Sentivist Server must be added to the system, and this provides automatic access to all those Sensors which have already been configured to report to that Server.

Further granularity of management is provided by the *Site*, which is used at the EC level to identify Sentivist Servers that are in the same physical location. This allows the administrator to sort alerts by location as well as by Server, and it is possible to create *Site Filters* to identify and ignore benign traffic from a particular Site.

The *Package Updater* tool is used to check for and import Package updates, which appear regularly on the NFR Package Server. If corporate security policy prohibits direct access to the NFR Package Server, updates can be downloaded separately and applied directly from local files.

After updated Packages have been imported, they (and their Backends) can be reviewed and applied or ignored as required. Whenever an update is applied to an installed Package or Backend, the policies for those Sentivist Servers and Sensors on which it is installed will also be updated automatically.

Managing database space is accomplished via the *Space Management* tool, which can be configured to delete automatically the oldest data from all or specific Backends as required.

Policy Management

Surprisingly, the concept of Policy has only been introduced to NFR IDS and IPS products with the recent introduction of the EC. However, taking its time in introducing this feature has allowed NFR to observe the competition and create an impressive Policy management system.

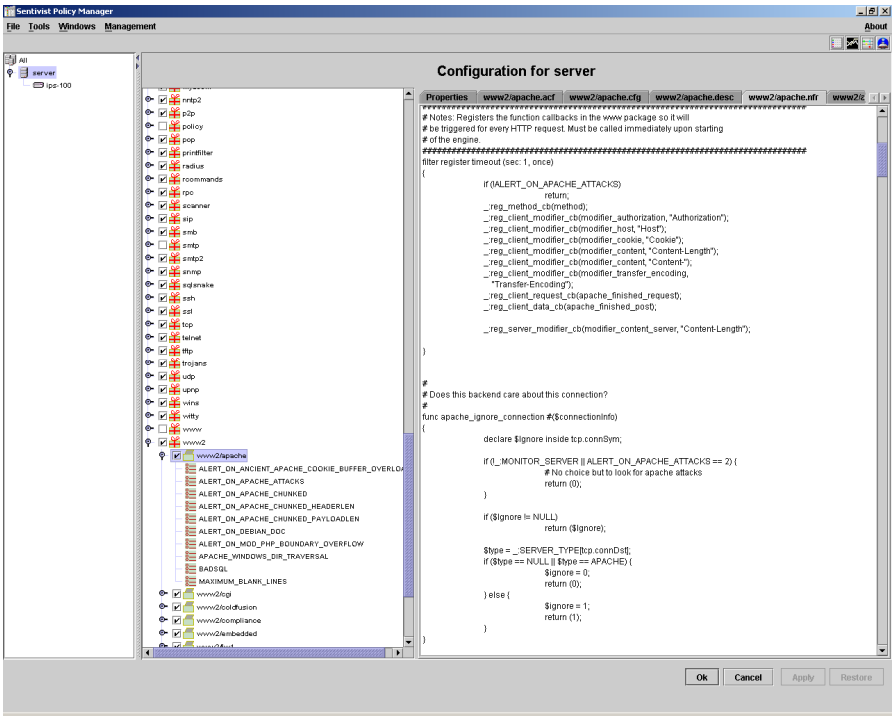

Figure 3 - Sentivist: Policy management

NFR has put some thought into making Policy definition as straightforward as possible. It needs to be, since Sentivist is a very complex and flexible system, and the way its signatures are arranged into *Packages* and *Backends*, with functionality modified as required by numerous, often cryptic, *Variables*, means that Policy creation from scratch is not as intuitive as with many other products.

A Sentivist Sensor monitors for network exploits based on the Packages and Backends it runs, and these contain the actual code (known as N–Code) that filters and processes events.

Each Package monitors the network for a specific category of exploit - the WWW Package, for example, filters Web server traffic. A Package also comprises one or more Backends that monitor the network for specific exploits.

WWW Package, for example, has one Backend that watches for attacks against Apache Web servers, and a separate one that watches for attacks against IIS Web servers.

A hierarchical tree display lists the Packages at the highest level, followed by the individual Backends, and finally the associated Variables. Selecting any element in the tree brings up a multi-tabbed display on the right where the administrator can choose to use the inherited value (more on inheritance later) or override with a value of his own.

Also in the tabbed display are tabs showing the various low-level configuration/description files for the Package or Backend in question, detailed information on what is covered by the Package/Backend, and even the exact N-Code, making this one of the most open systems available (the EC only allows the user to view N-Code, not modify it - N-Code may be added or modified via a separate text editor).

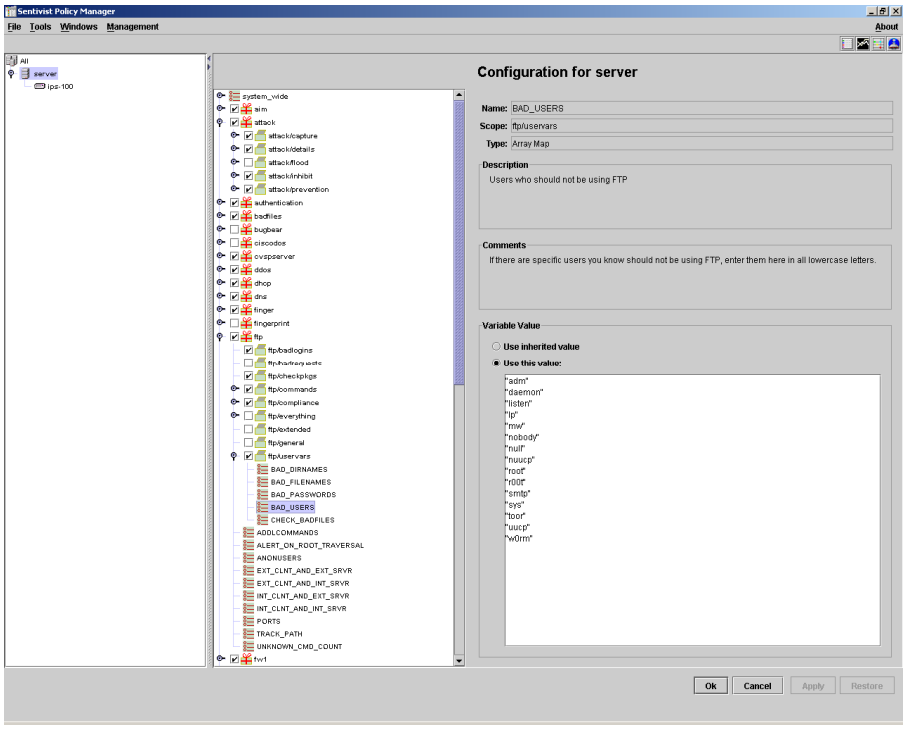

Figure 4 - Sentivist: Configuring Package variables

Entire Packages or individual Backends can be enabled/disabled, or installed to/uninstalled from the Sensor, using check boxes within the menu tree. Note also that all alert names, exploit descriptions and severity levels are “soft”, allowing the administrator to rename or redefine them as required.

Where the system departs intuitively from other IPS products is that individual “signatures” do not exist as such - in fact, each signature is made up of a snippet of N-Code, and multiple related signatures are rolled into a single Backend. Package and Backend behaviour is then affected by certain Variables that can be modified for each Sentivist Sensor.

Variables include IP addresses to ignore (whitelists) or always deny (blacklists), time periods to use for certain processes, maximum lengths of certain usernames, passwords, and commands, and so forth.

In some Backends, Variables can also be used to enable or disable individual “signatures” where this is deemed advantageous by the NFR code-writers. Thus, if you wished to disable the “*Nimda Scan*” signature, it may not be possible to do this via the enabling/disabling of Packages or Backends, or by modifying individual Variables. For this reason, there is no search facility within the Policy editor - there would be no point to it.

The power and flexibility to Sentivist Policies is provided by *Confidence Modifiers* and *Inhibit Rules*. Every alert raised by Sentivist has a *Confidence Level* associated with it. This is generated within the N-Code for each signature and is based on the confidence the signature writers have that the alert is accurate and immune to false positives. As the code progresses, what started out as a relatively low Confidence Level may be increased as new factors are determined from the traffic flow (a form of built-in low-level correlation mechanism).

A Variable in the *Attack/Prevention* Backend called *IPS Attacks* determines which alerts will be blocked when the appliance is in *prevention* mode based on the Confidence Level. Whole groups of signatures can have blocking enabled or disabled *en masse* by simply specifying the Confidence Level above which prevention (i.e. blocking the attack) should be enforced. Below the specified levels, an alert is still raised but the attack is no longer blocked.

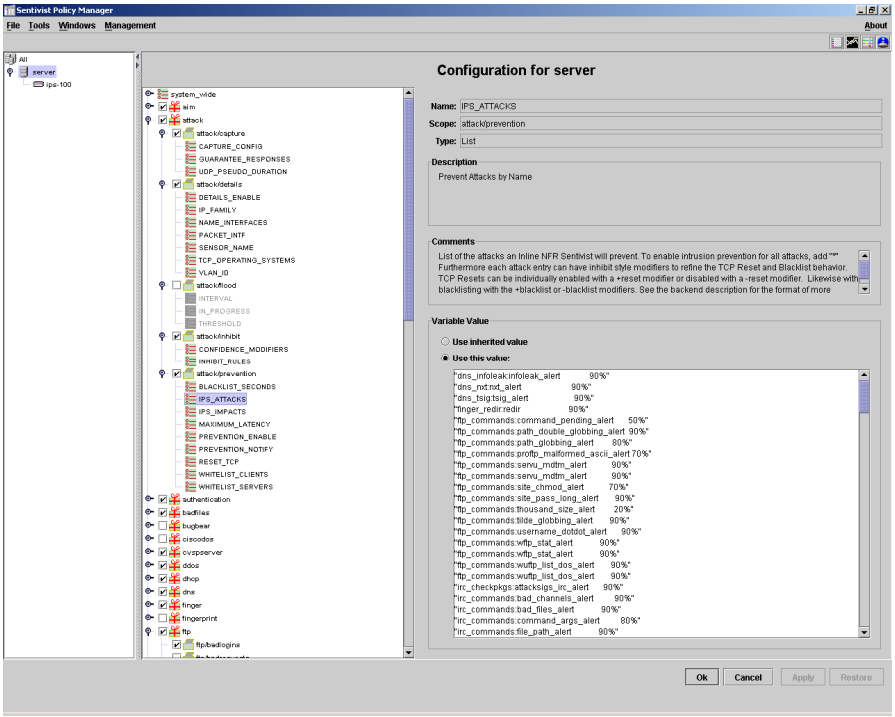

Figure 5 - Sentivist: Confidence Modifiers determine which attacks are blocked

This feature also allows administrators to set - again at a granular level or *en masse* - whether a particular type of attack will be mitigated via dropping the session or resetting the connection, and whether that type of attack will result in a temporary black-listing of the source IP.

Using the *Confidence Modifier* Variable, the administrator can override the levels set by the signature writers on a per-signature, per-Backend or per-Package basis, raising or lowering by a set amount as required.

Inhibit Rules can then be used to enable or disable recording, alerting and packet capturing (captures can be the trigger packet only, or a set number of packets from the same flow), also on a per-signature, per-Backend or per-Package basis.

Each Sentivist Sensor is therefore controlled by the following configuration settings:

- *Installed Packages and Backends*
- *The state of each Package and Backend (enabled or disabled)*
- *Configurable Variables associated with each Package and Backend*
- *User accounts and permission levels*

These configuration settings are known collectively as a "*policy*", and Enterprise Console allows these settings to be copied from one Sensor to another. Furthermore, policies can be applied to multiple Sensors at the same time, through the use of *Policy Groups*.

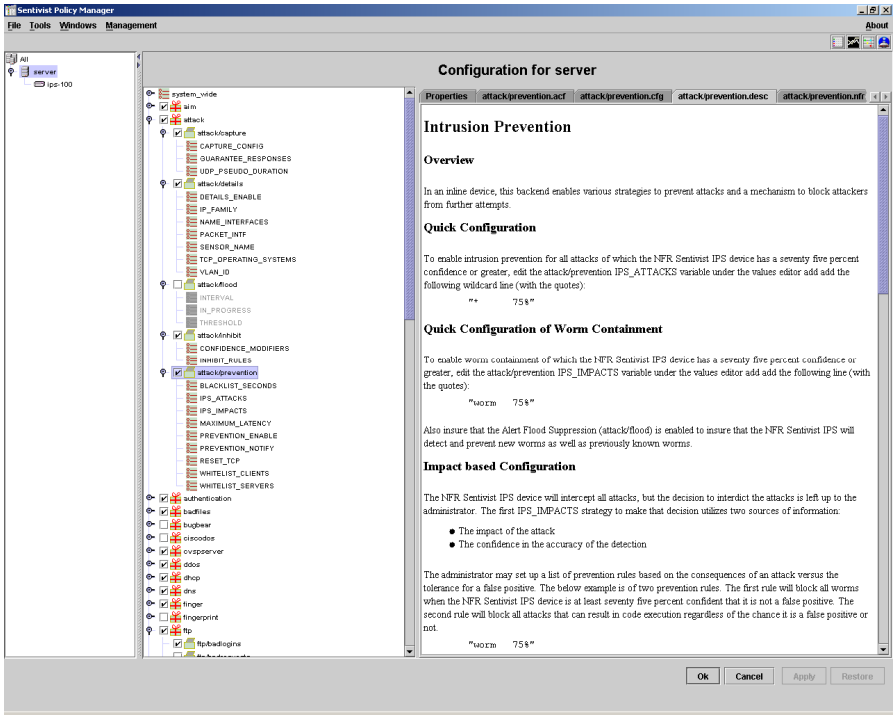

Figure 6 - Sentivist: Each Backend/Package includes extensive configuration information

A Policy Group is simply a collection of Sentivist Sensors, created for the purpose of applying policies. The group structure determines how the policies are applied, and EC automatically creates two levels of Policy Groups by default:

- *A top–level Policy Group called **All**, which encompasses all of the Sentivist Servers in the system. The All group is where the default settings for the system are stored*
- *A Policy Group corresponding to each individual Sentivist Server in the system, with All as its parent and the Sentivist Sensors attached to the Server as its children.*

Depending on requirements, many organisations may be able to manage policies using the default Groups. However, it is possible to create additional sub-Groups under any Server Group.

A typical use for this would be to create separate policies for DMZ servers, for Web servers, for critical internal servers, and so on.

Once Policy Groups have been created and separate policies defined for each Group, it is simply a matter of cutting and pasting Sensors into the appropriate Policy Group in order to apply the policy to one or more Sensors in a single operation. If at a later date a new policy needs to be applied to an individual Sensor, the Sensor need only be moved to the new Group for that to happen.

If the policy needs to be amended for all DMZ sensors, however, all that is necessary is to amend the policy at the DMZ Policy Group level, and it will automatically be rolled out to all Sensors within that Group. It is also possible to cut and paste policies between different Sensors and Policy Groups, if required.

Even at the point of policy application, the EC provides a list of all Servers and Sensors which will be affected, allowing the administrator to deselect any of them individually if he does not wish the new policy to be rolled out to specific Sensors. This is all very simple, and very effective - but it gets better.

By default, a "*child*" Policy Group automatically inherits values from its "*parent*". A Sentivist Server Policy Group, for example, inherits values from the *All* Group, and an individual Sensor inherits values from the Server Group (or Server sub-Group) to which it belongs. This is the mechanism by which the administrator can pass down the same configuration settings to multiple Sensors.

However, it is also possible to override these inherited configuration values by modifying the policy at a lower level - directly at the Sensor, for example. Where custom values have been defined at lower levels, they can subsequently be overridden again (effectively re-applying an upper-level policy configuration) by selecting a Server Group and clicking on the *Pass Down* option (which will re-apply the Server Group policy to **all** Sensors within that Group), or selecting an individual Sensor and clicking on the *Inherit* option (which will pull down the Group policy to that Sensor only). This system makes policy deployment a snip in even the largest installations.

We mentioned earlier in this section that Policy definition can be less than intuitive because of the complex nature of Sentivist. Whilst this may be true to some extent, the default settings provided by NFR are a good place to start, and will probably be adequate for most organisations.

The real trick is in getting to grips with which Package/Backend does what, and how the default behaviour can be modified, inhibited or overridden by clever use of the Variables. Once that has been mastered, and coupled with the hierarchical Policy Group and inheritance system, Sentivist offers one of the most powerful and flexible Policy management capabilities on the market today.

Alert Handling

The *Alert Browser* is the window that appears the first time the Enterprise Console is opened, and consists of four main areas:

- ***Alerts panel -*** *The alerts panel shows alerts received by the Enterprise Server in a continually-scrolling, near-real-time display. These alerts can be filtered to focus on certain criteria.*
- ***Held Alerts panel -*** *The administrator can drag specific alerts into the Held Alerts panel for further analysis away from the continually scrolling display. Held alerts do not automatically expire - they are held until the end of the current session.*
- ***Filter trees -*** *The two filter trees are used to filter the alerts that appear in the Alerts panel. The top filter tree is used to display only alerts that meet certain criteria. The bottom filter tree is used to hide (ignore) specific alerts or groups of alerts*
- ***Status summary -*** *This area in the lower left-hand corner of the window shows the number of alerts currently displayed, by priority (High, Medium, Low, System, Other)*

NFR has clearly put some thought into how the Alerts are configured out of the box, and the default configuration should suffice for most needs. Where changes are necessary, they are made easier by the use of *Alert Groups*.

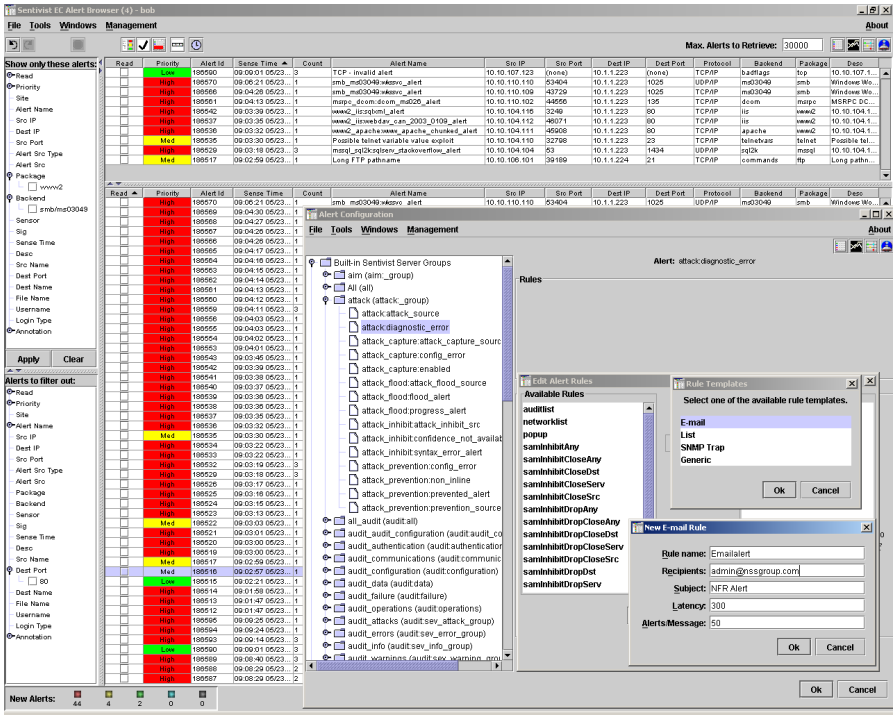

Figure 7 - Sentivist: Alert Groups

The behaviour of each alert type is dictated by the alert rules associated with it. The most simple alert rules tell Sentivist Server to record the alert to a list or display the alert in a pop-up window. For ease of management, similar alert types are organised into groups, and through these groups, the administrator can add alert rules to a set of alerts at one time, changing the way that all alerts in the group are processed.

Thus, the administrator could decide that all FTP-related alerts should be recorded only in the Recorder, whereas all DOS attacks should be recorded and appear in the EC Alert Browser window. If he subsequently decides that DOS attacks should also send an e-mail, it is a simple matter to add an e-mail rule to the DOS attacks Alert Group, and that action is effected for all alerts in that group immediately.

Sentivist Sensors provide the following response options:

- *Drop only*
- *Pass and Track*
- *Pass and Alert*
- *Alert only*
- *Drop packets*
- *Block TCP sessions*
- *Blacklisting of source IP addresses*
- *Capture packets (trigger packet only, or trigger and subsequent packets to a user-defined limit)*
- *Log event*
- *E-mail*
- *Pager*
- *SEMs (including ArcSight, Intellitactics, NetForensics, Symantec SESA among others).*
- *SNMP*
- *Syslog*

Sentivist also boasts *Dynamic Worm Mitigation*, utilising statistical alert behaviour to identify and automatically quarantine rapidly, self-propagating worms.

Applying Rules at group level makes sense in most circumstances, but NFR also allows the administrator to override the Alert Rules on a per-alert basis, if required. Configuration of alert responses has been well implemented in this system.

In the Alerts panel, alerts are displayed by Alert ID, with the most recent alerts at the top. By right-clicking on the column headings the administrator can choose the list of columns to display from all the data items stored against each alert in the database:

- ***Alert ID -*** *A unique identifier for the alert*
- ***Alert Name -*** *The name of the alert*
- ***Alert Src -*** *The name of the Sentivist Server that recorded the alert*
- ***Alert Src Type -*** *The source of the alert: Network, Cluster Correlator, Boolean Correlator, Firewall, or Host*
- ***Annotation -*** *Indicates whether an alert has comments*
- ***Backend -*** *The name of the backend that generated the alert*
- ***Count -*** *The number of duplicate alerts received by Enterprise Console within a two minute period. Duplicate alerts have the same signature, source IP address, destination IP address, and destination port.*
- ***Desc -*** *The alert description*
- ***Dest IP -*** *The destination IP address*
- ***Dest Name -*** *The destination IP name*
- ***Dest Port -*** *The destination port number*
- ***Package -*** *The name of the package that generated the alert*
- ***Priority -*** *The alert priority*
- ***Protocol -*** *The protocol used by the message that generated the alert*
- ***Read -*** *Indicates whether the alert is read*
- ***Rollup ID -*** *A unique identifier for duplicate alerts*
- ***Sense Time -*** *The time Sentivist Sensor generated the alert*
- ***Sensor -*** *The name of the Sentivist Sensor that generated the alert*

- ***Sig -*** *The signature of the alert (package, backend, and alert ID)*
- ***Site -*** *Short name of the Sentivist Server site*
- ***Src IP -*** *The source IP address*
- ***Src ID -*** *An identifier generated by Enterprise Console to uniquely identify each alert*
- ***Src Name -*** *The source name*
- ***Src Port -*** *The source port*
- ***Update DT -*** *The last time the alert record was updated in the Enterprise Console database (for example, the record is updated if the value of Count changes)*

Once the columns have been selected, they can be reordered by dragging them around the screen, and clicking on any column heading will re-sort the alerts by that column.

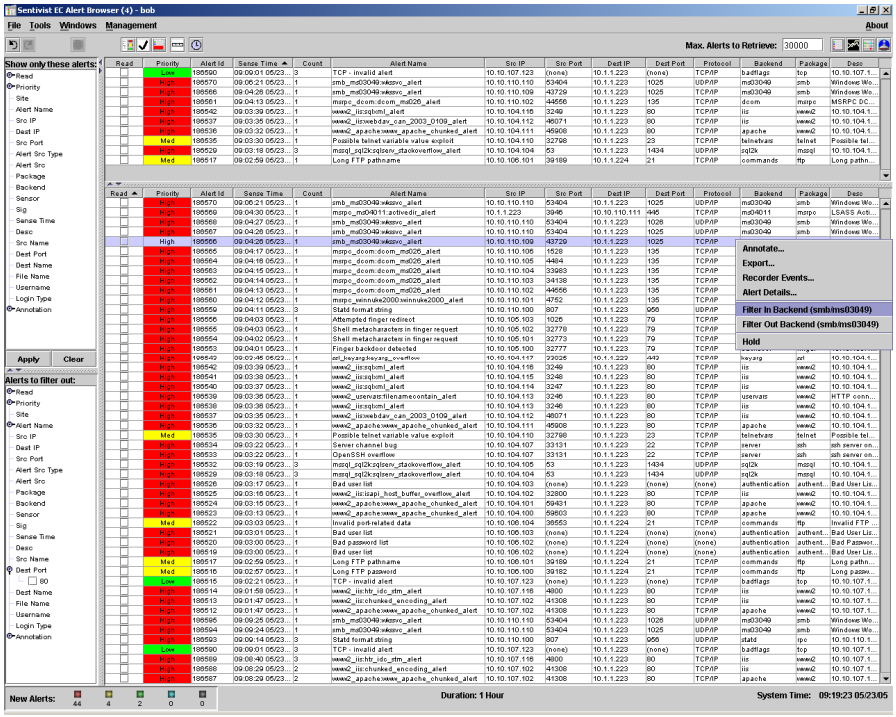

Figure 8 - Sentivist: Alert Browser

If the system receives multiple duplicate alerts at nearly the same time, it automatically combines them into a single alert - this can happen if multiple Sentivist Sensors detect the same incident all at once.) In this case, the number of combined alerts is shown in the *Count* column.

If the main alert pane is scrolling too fast, a *Cancel Load* button will pause the display to allow further analysis. Alerts can also be dragged into the *Held Alerts* pane where they can be saved until the end of the current session and analysed at leisure without scrolling. Once an alert or group of alerts has been dealt with or dismissed, they can be marked as read, causing them to disappear from the alert pane (although they naturally remain in the database for subsequent analysis and reporting).

Another toolbar button presents the current alert data in five separate panels, one for each priority level. The administrator can scroll through each panel individually.

By default, EC displays all alerts received by the Enterprise Server. However, the two filter trees on the left-hand side of the Alert Browser window can be used alone or together to display only the alerts required. The top filter tree ("*show only these alerts*") is used to display only those alerts which meet the specified criteria. For example, the administrator could specify that he only wishes to see alerts from one particular site, or only wants to see Web-based alerts.

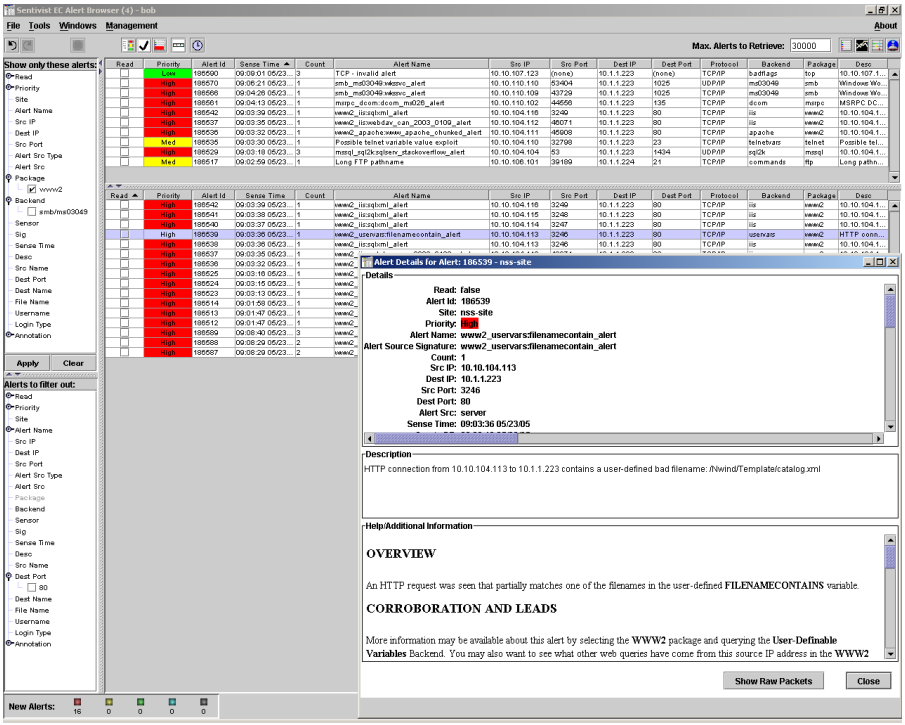

Figure 9 - Sentivist: Alert Filters in action

The bottom filter tree ("*alerts to filter out*") is used to specify those alerts that should be hidden or ignored. For example, the administrator could specify that he does not want to see alerts that have already been marked as *read*, or does not want to see alerts from the DMZ. The system will display all other alerts.

In either filter tree, right-clicking on an alert criteria brings up a pop-up dialogue box allowing the criteria to be entered by hand. A quicker way, however, is to right-click on any cell in the main alert pane and select the "*filter in*" or "*filter out*" menu option. For example, right-clicking on a particular IP address would allow the administrator to view *only* alerts from that address, or *exclude* all alerts from that address.

Multiple criteria can be set before clicking the *Apply* button, and once the administrator is happy with the view it can be saved - this saves all the column settings, layout and filter settings for subsequent recall. When recalling a previously-saved Alert Browser view, each one is opened in a fresh window, and thus as many different views can be open at one time as the administrator can handle (individual views can also be restricted to individual administrators if required). All the set criteria are shown as part of the filter tree, each with a check box against it.

Thus it is a simple matter for the administrator to select and deselect individual criteria and observe the effect on the alert display. The filtering capabilities are very simple to use, yet very powerful and useful.

Double-clicking on any alert brings up a dialogue box containing all of the alert details (as listed previously) including context data if available (such as the URI which caused the alert), a full description, and detailed help and reference information for the exploit (including impact, possible false positive conditions, CVE/Bugtraq references and remediation information). If packet capture was enabled for the Backend, raw packet data can be viewed within Ethereal by clicking on the *Show Raw Packets* button.

Demonstrating its pedigree in the IDS market place, Sentivist allows the administrator to *annotate* individual alerts (or groups of alerts), adding related alert IDs, reason for annotation, person responsible, and current status (under investigation, pending, closed, etc.). It is unfortunate that a built-in report is not provided to list outstanding/closed events.

The system can also consolidate alerts automatically using *correlators*. A correlator is an enterprise-level rule that combines multiple alerts that meet certain criteria, and is designed to help the administrator detect patterns among the alerts received.

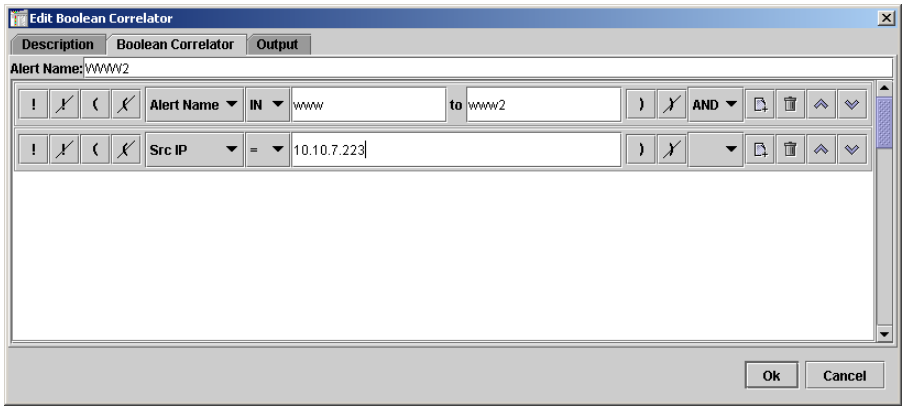

Figure 10 - Sentivist: Correlator rules

For example, Enterprise Console can be configured to send itself an additional alert if it receives fifty alerts from the same Source IP within two minutes. Two types of correlators are available:

- ***Cluster correlators*** *watch for alerts that contain identical values within specific fields – for example, all alerts that contain identical alert source signatures (regardless of the actual value)*
- ***Boolean correlators*** *watch for alerts that contain specific values within specific fields - for example, all alerts containing a specific source IP address and destination port.*

Annotation and correlation are the types of features which most “new” IPS vendors do not include, since the thinking is that once an alert has been blocked it is no longer necessary to investigate it, and correlation is irrelevant. Given that these devices can also be used in alert-only or passive IDS mode, however, we think these are features worth having.

In addition to the Alert Browser window, Enterprise Console provides the ability to view the *Alert History* (alerts which appeared within a user-specified time period), a *Timeline* window and three different types of real-time graphs:

- ***Activity Level Graph -*** *provides a view of the overall number of alerts being generated*

- ***Pick Graph -*** *allows the administrator to view activity level by a specific value or values (such as individual alert source IPs, destination ports, or alert names)*
- ***Top n Graph -*** *allows the administrator to examine a particular criterion by frequency (for example, the top three most-active alert source IPs).*

It is possible to print, save, and modify all of these graphs.

In the very useful *Timeline* display, alerts are plotted graphically on a moving “time line”, each represented by the corresponding priority colour. Scroll buttons at the top of the window can be used to move backward and forwards along the timeline, and the *Resolution* drop-down list can be used to set the time intervals, providing either a high-level view of the recent days/hours of activity, or a more focussed view on the last minute or ten seconds.

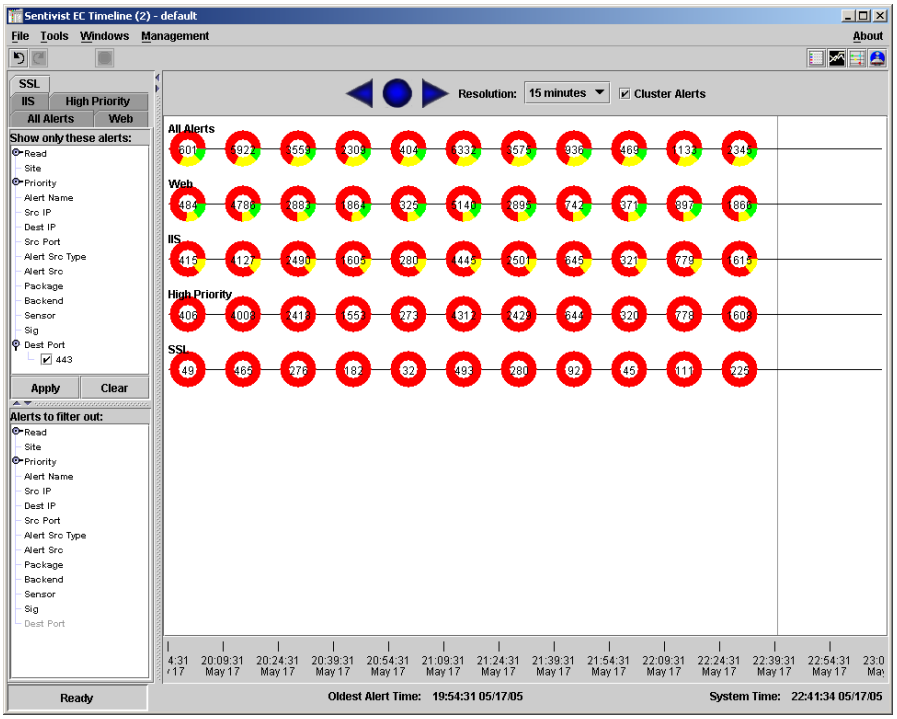

Figure 11 - Sentivist: Timeline display

When a larger period of time is being viewed, it is useful to be able to cluster the alerts, which replaces the confusing bunching of dots representing individual alerts with more readable miniature pie-charts, each section of the pie representing totals of alerts grouped by priority. Double clicking on any of the individual or clustered alerts launches a new Alert Browser window with the appropriate time filter in place to show just those alerts.

If only a single line of all alerts was available, this feature would be nothing more than a graphical gimmick. However, the administrator can create as many additional time lines as required, and a separate include/exclude filter can be applied to each line. Thus, it is possible to create a comprehensive real-time display of all alerts for specific groups of servers, for specific alerts, for inbound traffic only, for outbound traffic only, for Web traffic only, and so on.

When configured to monitor the most critical servers for the most serious alerts, the constantly scrolling graphical nature of this eye-catching display

makes it incredibly useful as a high-level immediate insight into what is happening on the network.

Reporting and Analysis

Most analysis will be performed using the wide range of alert handling tools described in the previous section. For those who wish to dig deeper into the alert data stored in the SQL repository, the schema is fully open.

In addition to the simple text-based *SQL Query* tool provided as part of EC, Sentivist provides a number of reports that can be generated through *Crystal Reports* from Business Objects (a separate Crystal Reports license must be purchased in order to do this, however).

Sentivist includes over 40 completely customisable Crystal Report Templates, including:

- *Top/Bottom Alerts*
- *Top/Bottom Source/Dest IPs*
- *Top/Bottom Source/Dest Ports*
- *Top/Bottom Services*
- *Top/Bottom Sensors*

A number of trending/comparison charts are also provided in the form of fully customisable Crystal Report Templates. These include:

- *Alerts by Date*
- *Alerts by Day of Week*
- *Alerts by Time of Day*
- *Alerts by Month*
- *Alerts by Service*
- *Alerts by Priority*

Naturally, any number of completely custom reports can also be created via Crystal Reports.

Verdict

Performance

The NFR Sentivist Smart Sensor 100C was tested up to 100Mbps, the rated speed of the device, and performance at all levels of our load tests was impeccable, with 100 per cent of all attacks being detected and blocked under all load conditions. We would happily confirm NFR's 100Mbps rating for this device, and would deem it conservative in most normal networks.

The basic latency figures of the NFR Smart Sensor 100C were excellent for a 100Mbps device across the board under all traffic loads. Behaviour throughout the tests with no background traffic was consistent and predictable, with small increases as additional network load was applied from 25Mbps to 100Mbps. HTTP response times were unusually high, however.

The device also had significant problems with our DDOS test, and would perform best when situated behind an attack mitigation device or a firewall with DDOS mitigation capabilities.

The Smart Sensor 100C performed consistently and completely reliably throughout our tests, blocking 100 per cent of attack traffic whilst passing 100 per cent of legitimate traffic, and exposing the sensor interface to ISIC-generated traffic had no long-term adverse effect on the device.

Security Effectiveness

Out of the box, signature recognition was good at 83 per cent, and was improved to 92 per cent following the application of a signature update after 48 hours. Blocking performance was identical following the update.

Of course, the real advantage of Sentivist is the fact that if there is a signature lacking, or one that does not work quite as you would expect, you can modify existing signatures or create custom ones using the N-Code programming language.

As with any programming language, N-Code requires time and effort to master, but as a “cross” between C++ and Perl it will feel instantly familiar to exponents of either, and anyone with any programming aptitude at all should be able to get to grips with it in a reasonable amount of time (training is available from NFR).

Whilst mastering N-Code is by no means essential to successful operation of the Sentivist system, it does make it one of the most customisable and extensible of all the products we have reviewed. Beware, however - such power can come at a price, since a poorly written piece of N-Code could significantly impact the performance of the sensor.

Performance in our “false negative” tests was adequate out of the box, and improved slightly following the signature update. The product also did very well at detecting most of the variants that we include alongside our basic test cases.

The device failed two test cases in our false positive test suite, and this was reduced to one (fake FireWall-1 RDP traffic) following the signature update. However, no other false positives were noted during stress testing with either Avalanche traffic or real-world traffic mixes.

Resistance to known evasion techniques was excellent, with the Smart Sensor 100C achieving a clean sweep across the board in most of our evasion tests. *Fragroute* and *Whisker* both failed to deceive the device into ignoring valid attacks, and all of the attempts were decoded accurately. Most of the miscellaneous evasion techniques (including extensive RPC record fragging) were handled well.

During testing, we verified the Smart Sensor 100C’s ability to handle up to 140,000 simultaneously open connections whilst successfully maintaining state on our half-open exploits. Unusually, stateless “exploits” are alerted upon by default, and blocked.

This is configurable, although our preferred configuration - suppress alerts and block mid-flows - was not possible (NFR state that it could be achieved via the use of additional N-Code and a filter, however).

Usability

Our main criticism of the management system comes down to cost. Since no management software is included at all with any Sensor, the minimum cost to be added to the Sensor price is $10,000 for the entry-level management license. On top of this, it is necessary to purchase a Crystal Reports license.

We would prefer to see some form of direct, single-sensor management capability included out of the box, and then an additional charge for managing multiple Sensors via a central console. It should also be possible to run at least some basic reports without having to purchase a full-blown Crystal Reports license. Given the extremely attractive prices of the Sensors themselves, this approach would offer a much more realistic entry-level cost for a basic system.

Having said that, there is not much left to criticise as far as management and usability goes. The three-tier management system is extremely powerful and scalable, and built-in management failover is provided via the ability to specify multiple Sentivist Servers for each Sensor to report to. The user roles are granular enough to suit most organisations, making the system very secure, and the Sensor can be deployed in passive IDS or in-line IPS (fail-open or fail-closed) modes.

Policy management has been extremely well thought out, with the ability to define multiple Policy Groups and drag and drop Sensors between those groups to apply policies. Additional flexibility is provided by the concept of *inheritance*, which allows certain parameters to be designed at a corporate or group level, whilst others are overridden on an individual Sensor basis.

Whilst policy definition may seem daunting initially, the use of Variables, Alert Groups, Confidence Levels and “modifiers” to override those Confidence Levels on a per-Package, per-Backend and per-Signature level makes this one of the most flexible and powerful policy definition systems we have seen. An added advantage for some organisations will undoubtedly be the use of N-Code for writing signatures, the ability to write your own Packages/Backends using that same language, and the fact the entire NFR library of Packages and Backends is completely open to inspection and modification.

Alert handling is similarly well-specified and well-implemented. The ability to launch multiple Alert Browser windows each with unique include/exclude filters, and to save them along with customised column layout, is very useful. Plenty of detail is available with each alert, including context information (where applicable) and raw packet captures. Filtering is fast and easy to do (either long hand or via right clicking on individual data items in the Alert Browser window), and the only feature missing at the moment is the ability to “roll up” groups of alerts into summary lines with totals - a feature which is planned for V5.0.

Reporting is well catered for too via numerous Crystal Reports templates, and the database schemas are completely open for those who wish to construct their own reports or feed them into third party event management/analysis systems.

All in all, both alert handling and reporting are extremely well-specified and well-implemented.

Contact Details

Company: NFR Security, Inc

E-mail: info@nfr.com

Internet: www.nfr.com

Address:
5 Choke Cherry Road
Suite 200
Rockville, MD 20850

Tel: +1 240 632 9000
Fax: +1 240 632 0200

APPENDIX A – TEST RESULTS

The aim of this procedure is to provide a thorough test of all the main components of an in-line Intrusion Prevention System (IPS) device in a controlled and repeatable manner and in the most “real world” environment that can be simulated in a test lab.

The Test Environment

The network is 100/1000Mbit Ethernet with CAT 5e cabling and Cisco 6500-Series switches (these have a mix of fibre and copper Gigabit interfaces). All devices are expected to be provided as appliances - if software-only, the supplier pre-installs the software on the recommended hardware platform. The sensor is configured as a perimeter device during testing (i.e. as if installed behind the main Internet gateway/firewall). There is no firewall protecting the target subnet.

Traffic generation equipment - such as the machines generating exploits, Spirent Avalanche and Spirent Smartbits *transmit* port - is connected to the “external” network, whilst the “receiving” equipment - such as the “target” hosts for the exploits, Spirent Reflector and Spirent Smartbits *receive* port - is connected to the internal network. The device under test is connected between two “gateway” switches - one at the edge of the external network, and one at the edge of the internal network.

All “normal” network traffic, background load traffic and exploit traffic will therefore be transmitted **through** the device under test, from external to internal. The same traffic is mirrored to a single SPAN port of the external gateway switch, to which an Adtech network monitoring device is connected. The Adtech AX/4000 monitors the same mirrored traffic to ensure that the total amount of traffic never exceeds 1Gbps (which would invalidate the test run).

The management interface is used to connect the appliance to the management console on a private subnet. This ensures that the sensor and console can communicate even when the target subnet is subjected to heavy loads, in addition to preventing attacks on the console itself.

Section 1 – Detection Engine

The aim of this section is to verify that the sensor is capable of detecting and blocking a wide range of common exploits accurately, whilst remaining resistant to false positives. All tests in this section are completed with **no background network load**. The latest signature pack is acquired from the vendor, and sensors are deployed with **all** available attack signatures enabled (some audit/informational signatures may be disabled).

Test 1.1 - Attack Recognition

Whilst it is not possible to validate completely the entire signature set of any sensor, this test attempts to demonstrate how accurately the sensor detects and blocks a wide range of common exploits, port scans, and Denial of Service attempts. These are updated/changed for every new test, and all exploits are run with no load on the network and no IP fragmentation.

Our attack suite contains over 100 basic exploits (plus variants) covering the following areas:

- ***Test 1.1.1 -*** *Backdoors (standard ports and random ports)*
- ***Test 1.1.2 -*** *DNS/WINS*
- ***Test 1.1.3 -*** *DOS*
- ***Test 1.1.4 -*** *False negatives (common exploits which have been modified to remove or alter obvious "triggers" - this ensures that the signatures are coded for the underlying vulnerability rather than a particular exploit)*
- ***Test 1.1.5 -*** *Finger*
- ***Test 1.1.6 -*** *FTP*
- ***Test 1.1.7 -*** *HTTP*
- ***Test 1.1.8 -*** *ICMP (including unsolicited ICMP response)*
- ***Test 1.1.9 -*** *Reconnaissance*
- ***Test 1.1.10 -*** *RPC*
- ***Test 1.1.11 -*** *SSH*
- ***Test 1.1.12 -*** *Telnet*
- ***Test 1.1.13 -*** *Database*
- ***Test 1.1.14 -*** *Mail*
- ***Test 1.1.15 -*** *Voice*

A wide range of vulnerable target operating systems and applications are used, and the majority of the attacks are successful, gaining root shell or administrator privileges on the target machine.

We expect all the attacks to be reported in as straightforward and clear a manner as possible (i.e. an "RDS MDAC attack" should be reported as such, rather than a "Generic IIS Attack"). Wherever possible, attacks should be identified by their assigned CVE reference. It will also be noted when a response to an exploit is considered too "noisy", generating multiple similar or identical alerts for the same attack. Finally, we will note whether the device blocks the attack packet only or the entire "suspicious" TCP session.

This test is repeated twice: the first run with blocking disabled on the sensor (monitor mode only) in order to determine which attacks are detected and how accurately they are detected (*Attack Recognition Rating*); the second run with blocking enabled in order to determine which attacks are blocked successfully regardless of how they are detected or what alerts are raised (*Attack Blocking Rating*)

The "**default**" *Attack Recognition Rating-Detect Only* (ARRD) and *Attack Recognition Rating-Block* (ARRB) are each expressed as a percentage of detected/blocked exploits against total number of exploits launched with the default signature set as received by NSS. This demonstrates how effective the sensor can be when simply deploying the default configuration.

Following the initial test run, each vendor is provided with a list of CVE references of the attacks missed, and is then allowed 48 hours to produce an updated signature set. This updated signature set **must** be released to the general public as a standard signature/product update before the report is published - this ensures that vendors do not attempt to code signatures just for this test.

The sensor is then exposed to a second round of identical tests and the "**custom**" ARRD/ARRB is determined. This demonstrates how effective the vendor is at responding to a requirement for new or updated signatures.

Both the *default* and *custom* ARRD/ARRB figures are reported.

Test 1.2 - Resistance To False Positives

The aim of this test is to demonstrate how likely it is that a sensor raises a false positive alert - particularly critical for IPS devices.

We have a number of trace files of normal traffic with "suspicious" content, together with several "neutered" exploits which have been rendered completely ineffective. If a signature has been coded for a specific piece of exploit code rather then the underlying vulnerability, or if it relies purely on pattern matching, some of these false alarms could be alerted upon.

The product attains a "PASS" for each test case if it does **not** raise an alert and does **not** block the traffic. Raising an alert on any of these test cases is considered a "FAIL", since none of the "exploits" used in this test represents a genuine threat. A "FAIL" would thus indicate the chance that the sensor could block legitimate traffic inadvertently.

- ***Test 1.2.1 -*** *False positives*

Section 2 – Evasion

The aim of this section is to verify that the sensor is capable of detecting and blocking basic exploits when subjected to varying common evasion techniques.

Test 2.1 - Baselines

The aim of this test is to establish that the sensor is capable of detecting and blocking a number of common basic attacks (our baseline suite) in their normal state, with no evasion techniques applied. Note that common/older attacks have been chosen deliberately for this particular test to ensure that ALL products tested have signatures in place for the evasion tests.

- ***Test 2.1.1 -*** *Baseline attack replay*

Test 2.2 - Packet Fragmentation and Stream Segmentation

The baseline HTTP attacks are repeated, running them through fragroute using various evasion techniques, including:

- ***Test 2.2.1 -*** *IP fragmentation - ordered 8 byte fragments*
- ***Test 2.2.2 -*** *IP fragmentation - ordered 24 byte fragments*
- ***Test 2.2.3 -*** *IP fragmentation - out of order 8 byte fragments*
- ***Test 2.2.4 -*** *IP fragmentation - ordered 8 byte fragments, duplicate last packet*
- ***Test 2.2.5 -*** *IP fragmentation - out of order 8 byte fragments, duplicate last packet*
- ***Test 2.2.6 -*** *IP fragmentation - ordered 8 byte fragments, reorder fragments in reverse*

- ***Test 2.2.7 -*** *IP fragmentation - ordered 16 byte fragments, fragment overlap (favour new)*
- ***Test 2.2.8 -*** *IP fragmentation - ordered 16 byte fragments, fragment overlap (favour old)*
- ***Test 2.2.9 -*** *TCP segmentation - ordered 1 byte segments, interleaved duplicate segments with invalid TCP checksums*
- ***Test 2.2.10 -*** *TCP segmentation - ordered 1 byte segments, interleaved duplicate segments with null TCP control flags*
- ***Test 2.2.11 -*** *TCP segmentation - ordered 1 byte segments, interleaved duplicate segments with requests to resync sequence numbers mid-stream*
- ***Test 2.2.12 -*** *TCP segmentation - ordered 1 byte segments, duplicate last packet*
- ***Test 2.2.13 -*** *TCP segmentation - ordered 2 byte segments, segment overlap (favour new)*
- ***Test 2.2.14 -*** *TCP segmentation - ordered 1 byte segments, interleaved duplicate segments with out-of-window sequence numbers*
- ***Test 2.2.15 -*** *TCP segmentation - out of order 1 byte segments*
- ***Test 2.2.16 -*** *TCP segmentation - out of order 1 byte segments, interleaved duplicate segments with faked retransmits*
- ***Test 2.2.17 -*** *TCP segmentation - ordered 1 byte segments, segment overlap (favour new)*
- ***Test 2.2.18 -*** *TCP segmentation - out of order 1 byte segments, PAWS elimination (interleaved dup segs with older TCP timestamp options)*
- ***Test 2.2.19 -*** *IP fragmentation - out of order 8 byte fragments, interleaved duplicate packets scheduled for later delivery*
- ***Test 2.2.20 -*** *TCP segmentation - ordered 16 byte segments, segment overlap (favour new (Unix))*

For each of the evasion techniques, we note if (i) the attempted attack is blocked successfully (the primary aim of any IPS device), (ii) the attempted attack is detected and an alert raised in **any** form, and (iii) if the exploit is successfully “decoded” to provide an accurate alert relating to the original exploit, rather than alerting purely on anomalous traffic detected as a result of the evasion technique itself.

Test 2.3 - URL Obfuscation

The baseline HTTP attacks are repeated, this time applying various URL obfuscation techniques made popular by the Whisker Web server vulnerability scanner, including:

- ***Test 2.3.1 -*** *URL encoding*
- ***Test 2.3.2 -*** */./ directory insertion*
- ***Test 2.3.3 -*** *Premature URL ending*
- ***Test 2.3.4 -*** *Long URL*
- ***Test 2.3.5 -*** *Fake parameter*
- ***Test 2.3.6 -*** *TAB separation*
- ***Test 2.3.7 -*** *Case sensitivity*
- ***Test 2.3.8 -*** *Windows \ delimiter*
- ***Test 2.3.9 -*** *Session splicing*

For each of the evasion techniques, we note if (i) the attempted attack is blocked successfully, (ii) the attempted attack is detected and an alert raised in **any** form, and (iii) if the exploit is successfully "decoded" to provide an accurate alert relating to the original exploit, rather than alerting purely on anomalous traffic detected as a result of the evasion technique itself.

Test 2.4 - Miscellaneous Evasion Techniques

Certain baseline attacks are repeated, and are subjected to various protocol- or exploit-specific evasion techniques, including:

- ***Test 2.4.1 -*** *Altering default ports/passwords for backdoors*
- ***Test 2.4.2 -*** *Inserting spaces in FTP command lines*
- ***Test 2.4.3 -*** *Inserting non-text Telnet opcodes in FTP data stream*
- ***Test 2.4.4 -*** *Polymorphic mutation (ADMmutate)*
- ***Test 2.4.5 -*** *Altering protocol and RPC PROC numbers*
- ***Test 2.4.6 -*** *RPC record fragging (MS-RPC and Sun)*
- ***Test 2.4.7 -*** *HTTP exploits to non-standard port*

For each of the evasion techniques, we note if (i) the attempted attack is blocked successfully, (ii) the attempted attack is detected and an alert raised in **any** form, and (iii) if the exploit is successfully "decoded" to provide an accurate alert relating to the original exploit, rather than alerting purely on anomalous traffic detected as a result of the evasion technique itself.

Section 3 – Stateful Operation

The aim of this section is to be able to determine whether the sensor is capable of monitoring stateful sessions established through the device at various traffic loads without either losing state or incorrectly inferring state.

Test 3.1 - Stateless Attack Replay (Mid-Flows)

This test determines whether the sensor is resistant to stateless attack flooding tools - these utilities are used to generate large numbers of false alerts on the protected subnet using valid source and destination addresses and a range of protocols.

The main characteristic of many flooding tools is the fact that they generate single packets containing "trigger" patterns without first attempting to establish a connection with the target server. Whilst this can be effective in raising alerts with some stateless protocols such as UDP and ICMP, they should never be capable of raising an alert for exploits based on stateful protocols such as FTP and HTTP.

In this test, we transmit a number of packets taken from capture files of valid exploits, but without first establishing a valid session with the target server. We also remove the session tear down and acknowledgement packets so that the sensor can not "infer" that a valid connection was made.

In order to receive a "PASS" in this test, no alerts should be raised for any of the actual exploits (although "mid-flow" alerts are permitted).

However, each packet should be blocked if possible since it represents a "broken" or "incomplete" session.

- ***Test 3.1.1 -*** *Stateless attack replay*

Test 3.2 - Simultaneous Open Connections (default settings)

This test determines whether the sensor is capable of preserving state across increasing numbers of open connections, as well as continuing to detect and block new exploits when the state tables are filled. It also attempts to determine whether or not the sensor will block legitimate traffic once state tables are filled. This test is run using the default sensor settings (no tuning of sensor parameters).

A legitimate HTTP session is opened and the first packet of a two-packet exploit is transmitted. The Spirent Avalanche (on the "external" interface of the sensor) then opens various numbers of TCP sessions from 10,000 to 1,000,000 (one million) with the Spirent Reflector (on the "internal" interface of the sensor). The initial HTTP session is then completed with the second half of the exploit and the session is closed. If the sensor is still maintaining state on the first session established, the exploit will be recorded. If the state tables have been exhausted, the exploit string will be seen as a non-stateful attack, and will thus be ignored.

Both halves of the exploit are required to trigger an alert - a product will fail the test if it fails to generate an alert after the second packet is transmitted, or if it raises an alert on either half of the exploit on its own.

At each step, we ensure that the sensor is still capable of detecting and blocking freshly-launched exploits once all the connections are open, as well as confirming that the device does not block legitimate traffic (perhaps as a result of state tables filling up). We then launch further exploits whilst the Avalanche/Reflector devices "churn" connections at the maximum level set, ensuring that the sensor is still capable of detecting and blocking freshly-launched exploits as old connections are torn down and new ones recreated constantly.

- ***Test 3.2.1 - Attack Detection:*** *This test ensures that the sensor continues to detect new exploits as the number of open sessions is increased in stages from 10,000 to 1,000,000*
- ***Test 3.2.2 - Attack Blocking:*** *This test ensures that the sensor continues to block new exploits as the number of open sessions is increased in stages from 10,000 to 1,000,000*
- ***Test 3.2.3 - State Preservation:*** *This test ensures that the sensor maintains the state of pre-existing sessions as the number of open sessions is increased in stages from 10,000 to 1,000,000*
- ***Test 3.2.4 - Legitimate Traffic Blocking:*** *This test ensures that the sensor does not begin to block legitimate traffic as the number of open sessions is increased in stages from 10,000 to 1,000,000*

Test 3.3 - Simultaneous Open Connections (after tuning)

Test 3.2 is repeated after any tuning recommended by the vendor (if applicable) to increase the size of the state tables.

- ***Test 3.3.1 - Attack Detection:*** *As Test 3.2.1 following tuning*
- ***Test 3.3.2 - Attack Blocking:*** *As Test 3.2.2 following tuning*
- ***Test 3.3.3 - State Preservation:*** *As Test 3.2.3 following tuning*
- ***Test 3.3.4 - Legitimate Traffic Blocking:*** *As Test 3.2.4 following tuning*

Section 4 – Detection/Blocking Performance Under Load

The aim of this section is to verify that the sensor is capable of detecting and blocking exploits when subjected to increasing loads of background traffic up to the maximum bandwidth supported as claimed by the vendor.

The latest signature pack is acquired from the vendor, and sensors are deployed with **all** available attack signatures enabled (some audit/informational signatures may be disabled). Each sensor is configured to **detect and block** suspicious traffic.

Our “attacker” host launches a fixed number of exploits at a target host on the subnet being protected by the device under test. The Adtech network monitor is configured to monitor the switch SPAN port consisting of normal, exploit and background traffic, and is capable of reporting the total number of exploit packets seen on the wire as verification.

A fixed number of exploits are launched with zero background traffic to ensure the sensor is capable of detecting our baseline attacks. Once that has been established, increasing levels of varying types of background traffic are generated **through** the sensor in order to determine the point at which the sensor begins to miss attacks - all tests are repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic (or up to the maximum rated throughput of the device should this be less than 1Gbps).

At all stages, the Adtech network monitor verifies both the overall traffic loading and the total number of exploits seen on the target subnet. An additional confirmation is provided by the target host which reports the number of exploits which actually made it through.

The *Attack Blocking Rate* (ABR) at each background load is expressed as a percentage of the number of exploits blocked by the sensor (when in blocking mode) against the number verified by the Adtech network monitor and target host. The *Attack Detection Rate* (ADR) at each background load is expressed as a percentage of the number of exploits detected by the sensor (with blocking mode disabled) against the number verified by the Adtech network monitor and target host.

For each type of background traffic, we also determine the maximum load the sensor can sustain before it begins to drop packets/miss alerts. It is worth noting that devices which demonstrate 100 per cent ABR (blocking) but less than 100 per cent ADR (detection) in these tests will be prone to blocking **legitimate** traffic under similar loads.

Test 4.1 - UDP Traffic To Random Valid Ports

This test uses UDP packets of varying sizes generated by a **Smartbits SMB6000** with LAN-3301A 10/100/1000Mbps **TeraMetrics** cards installed.

A constant stream of the appropriate mix of packets - with variable source IP addresses and ports transmitting to a single fixed IP address/port - is transmitted through the sensor (bi-directionally, maximum of 1Gbps).

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 packets per second (pps) figures are verified by the Adtech Gigabit network monitoring tool before each test begins. Multiple tests are run and averages taken where necessary.

This traffic does not attempt to simulate any form of “real world” network condition. The aim of this test is purely to determine the raw packet processing capability of the sensor, and its effectiveness at passing “useless” packets quickly in order to pass potential attack packets to the detection engine. The range of packet sizes has been selected to mirror the maximum, minimum and average packet sizes used in our HTTP stress tests.

- ***Test 4.1.1 - 256 byte packets - maximum 453,000 packets per second:*** *This test is roughly equivalent to a 40,000 connections per second test in our HTTP stress tests (in terms of packet size and packets per second rate), and has been included to provide an indication of the packet processing performance under the most extreme conditions for most devices - it is unlikely that any real-life network will ever see network loads of over 450,000 256-byte packets per second unless under severe DOS conditions. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic.*
- ***Test 4.1.2 - 550 byte packets - maximum 220,000 packets per second:*** *This test has been included to provide a comparison with our “real world” packet mixes, since the average packet size is similar. No sessions are created during this test and there is very little for the detection engine to do in the way of protocol analysis. This test provides a reasonable indication of the ability of a device to process packets from the wire on an “average” network, and we would expect all products to demonstrate good performance levels. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic.*
- ***Test 4.1.3 - 1000 byte packets - maximum 122,000 packets per second:*** *This test is the complete opposite of the 256 byte packet test, in that we would expect every single product to be capable of returning 100 per cent detection rates across the board when using only 1000 byte packets. We have included this test mainly to demonstrate how easy it is to achieve good results using large packets – beware of test results that* ***only*** *quote performance figures using similar (or larger) packet sizes. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic.*

Test 4.2 - HTTP “Maximum Stress” Traffic With No Transaction Delays

HTTP is the most widely used protocol in most normal networks, as well as being one of the most widely exploited. The number of potential HTTP exploits for the protocol makes a pure HTTP network something of a torture test for the average sensor.

The use of multiple Spirent Communications **Avalanche 2500** and **Reflector 2500** devices allows us to create true “real world” traffic at speeds of up to 4.2 Gbps as a background load for our tests. Our Avalanche configuration is capable of simulating over 5 million users, with over 5 million concurrent sessions, and over 200,000 HTTP requests per second.

By creating genuine session-based traffic with varying session lengths, the sensor is forced to track valid 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” as it is possible to achieve in a lab environment, whilst ensuring absolute accuracy and repeatability.

The aim of this test is to stress the HTTP detection engine and determine how the sensor copes with detecting and blocking exploits under network loads of varying average packet size and varying connections per second.

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

- ***Test 4.2.1 -*** *Max 2,500 new connections per second - average packet size 1000 bytes - maximum 120,000 packets per second. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic. With relatively low connection rates and large packet sizes, we expect all sensors to achieve 100% blocking rates throughout this test.*
- ***Test 4.2.2 -*** *Max 5,000 new connections per second - average packet size 540 bytes - maximum 225,000 packets per second. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic. With average connection rates average packet sizes, this is a good approximation of a real-world production network, and we expect all sensors to perform well in this test.*
- ***Test 4.2.3 -*** *Max 10,000 new connections per second - average packet size 440 bytes - maximum 275,000 packets per second. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic. With average packet sizes coupled with very high connection rates, this is a strenuous test for any sensor, and represents a very heavily used production network.*
- ***Test 4.2.4 -*** *Max 20,000 new connections per second - average packet size 360 bytes - maximum 320,000 packets per second. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic. With small packet sizes and extremely high connection rates this is an extreme test for any sensor. Not many sensors will perform well at all levels of this test.*

Test 4.3 - HTTP “Maximum Stress” Traffic With Transaction Delays

This test is identical to Test 4.2 except that we introduce a 10 second delay in the server response for each transaction. This has the effect of maintaining a high number of open connections throughout the test, thus forcing the sensor to utilise additional resources to track those connections.

- ***Test 4.3.1 -*** *Max 5,000 new connections per second - average packet size 540 bytes - maximum 225,000 packets per second - 10 second transaction delay - maximum 50,000 open connections. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic. With average connection rates average packet sizes, this is a good approximation of a real-world production network, and we expect all sensors to perform well in this test.*
- ***Test 4.3.2 -*** *Max 10,000 new connections per second - average packet size 440 bytes - maximum 275,000 packets per second - 10 second transaction delay - maximum 100,000 open connections. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic. With average packet sizes coupled with very high connection rates, this is a strenuous test for any sensor, and represents a very heavily used production network.*

Test 4.4 - Protocol Mix Traffic

Whereas 4.2 and 4.3 provide a pure HTTP environment with varying connection rates and average packet sizes, the aim of this test is to simulate more of a “real world” environment by introducing additional protocols whilst still maintaining a precisely repeatable and consistent background traffic load (something rarely seen in a real world environment).

The result is a background traffic load that, whilst less stressful than previous tests, is closer to what may be found on a heavily-utilised “normal” production network.

- ***Test 4.4.1 -*** *72% HTTP traffic (540 byte packets) + 20% FTP traffic + 6% UDP traffic (256 byte packets). Max 4000 new connections per second - average packet size 540 bytes - maximum 215,000 packets per second - maximum 750 open connections. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic. With lower connection rates, average packets sizes and a common protocol mix, this is a good approximation of a heavily-used production network, and we expect all sensors to perform well throughout this test.*

Test 4.5 - “Real World” Traffic

This is as close as it is possible to come to a true “real world” environment under lab conditions. For this test we eliminate the Reflector device and substitute an IIS Web server installed on a dual-Xeon server with Gigabit interface and 4GB RAM. This server holds a copy of The NSS Group Web site, and is capable of handling a full 1Gbps of traffic. We then capture a typical client browsing session on the NSS Group Web site, accessing a mixture of menu pages, lengthy text-based reports and multiple graphical images (screen shots) and have Avalanche replay multiple identical sessions from up to **20 new users per second**.

It should be noted that whereas the goal of the previous tests is a very predictable, consistent and repeatable background load that never varies, the nature of this test means that traffic is slightly more “bursty” in nature.

- ***Test 4.5.1 - Pure HTTP Traffic (simulated browsing session on NSS Web site):*** *Max 4700 new connections per second - 20 new users per second - average packet size 560 bytes - maximum 210,000 packets per second.*

*Repeated with 250Mbps, 500Mbps, 750Mbps and 950Mbps of background traffic. With genuine server responses to genuine **browser** sessions consisting **of multiple transactions per** session, this is a typical "real world" background load, albeit pure HTTP. Although the Web server and the network are extremely busy at the higher traffic loads, the "normal" connection rates and packet sizes should enable most sensors to perform well at all load levels in this test.*

- ***Test 4.5.2 - Protocol Mix (72% HTTP traffic (simulated browsing sessions as 4.5.1)) + 20% FTP traffic + 6% UDP traffic (256 byte packets)):** Max 3700 new connections per second - average packet size 560 bytes - maximum 205,000 packets per second - maximum 1,500 open connections.*

  *Repeated with 250Mbps, 500Mbps, 750Mbps and 950Mbps of background traffic. With genuine server responses to genuine browser sessions consisting of multiple **transactions** per session, mixed with FTP and UDP traffic, this is a typical "real world" background load. Although the Web server and the network are extremely busy at the higher traffic loads, the "normal" connection rates and packet sizes should enable most sensors to perform well at all load levels in this test.*

To gauge the effects of varying (smaller) packet sizes, connection rates and transaction delays, the results of tests 4.2 - 4.4 should be examined.

Section 5 – Latency & User Response Times

The aim of this section is to determine the effect the sensor has on the traffic passing through it under various load conditions.

Should a device impose a high degree of latency on the packets passing through it, a network or security administrator would need to think carefully about how many devices could be installed in a single data path before user response times became unacceptable or the combination of devices caused excessive timeouts. We also determine the effect of high levels of normal HTTP traffic and a basic DOS attack on the average latency and user response times.

Test 5.1 - Latency

We use Spirent SmartFlow software and The Smartbits SMB6000 with Gigabit TeraMetrics cards to create multiple traffic flows through the appliance and measure the basic throughput, packet loss, and latency through the sensor. This test - whilst not indicative of real-life network traffic - provides an indication of how much the sensor affects the traffic flow through it. This data is particularly useful for network administrators who need to gauge the effect of any form of in-line device which is likely to be placed at critical points within the corporate network.

SmartFlow runs through several iterations of the test varying the traffic load from 250Mbps to 1Gbps bi-directionally (or up to the maximum rated throughput of the device should this be less than 1Gbps) in steps of 250Mbps. This is repeated for a range of packet sizes (256 bytes, 550 bytes and 1000 bytes) of UDP traffic with variable IP addresses and ports. At each iteration of the test, SmartFlow records the number of packets dropped, together with average and maximum latency.

- ***Test 5.1.1 - Latency With No Background Traffic:*** *SmartFlow traffic is passed across the infrastructure switches and through the device (the latency of the basic infrastructure is known and is constant throughout the tests). The packet loss and average latency are recorded at each packet size and each load level from 250Mbps to 1Gbps (in 250Mbps steps).*
- ***Test 5.1.2 - Latency With Background Traffic Load:*** *The Avalanche and Reflector are configured to generate a fixed amount of background HTTP traffic through the sensor (up to 50 per cent of the maximum rated bandwidth of the device under test - maximum 500Mbps - maximum 2,500 new connections per second - average packet size 540 bytes - maximum 112,500 packets per second).*

  A 250Mbps bi-directional load of SmartFlow traffic at various packet sizes (256 bytes, 540 bytes and 1000 bytes) is then passed across the infrastructure switches and through the device and the packet loss and average latency are recorded.
- ***Test 5.1.3 - Latency When Under Attack:*** *The Spirent WebSuite software is used to generate a fixed load of DOS/DDOS traffic of 10 per cent of the maximum rated bandwidth of the device under test (maximum 100Mbps). A 250Mbps bi-directional load of SmartFlow traffic at various packet sizes (256 bytes, 540 bytes and 1000 bytes) is then passed across the infrastructure switches and through the device and the packet loss and average latency are recorded. The device should be configured to detect/block/mitigate the DOS attack by the most efficient method available.*

Test 5.2 - User Response Times

Avalanche and Reflector devices are used to generate HTTP sessions through the device in order to gauge how any increases in latency will impact the user experience in terms of failed connections and increased Web response times.

- ***Test 5.2.1 - Web Response With No Background Traffic:*** *The Avalanche and Reflector are configured to generate HTTP traffic through the sensor (up to 50 per cent of the maximum rated bandwidth of the device under test - maximum 500Mbps - maximum 2,500 new connections per second - average packet size 540 bytes - maximum 112,500 packets per second).*

  The minimum, maximum and average page response times and number of failed connections are recorded by Avalanche to provide an indication of the expected response times under normal traffic conditions.
- ***Test 5.2.2 - Web Response When Under Attack:*** *The Avalanche and Reflector are configured to generate HTTP traffic through the sensor as for Test 5.2.1. The Spirent WebSuite software is then used to generate DOS/DDOS traffic up to 10 per cent of the maximum rated bandwidth of the device under test (maximum 100Mbps).*

  The minimum, maximum and average page response times and number of failed connections are recorded by Avalanche to provide an indication of the expected response times when the device is under attack.

Section 6 – Stability & Reliability

These tests attempt to verify the stability of the device under test under various extreme conditions. Long term stability is particularly important for an in-line IPS device, where failure can produce network outages.

- ***Test 6.1.1 - Blocking Under Extended Attack:*** *For this test, we expose the external interface of the device to a constant stream of alerts over an extended period of time. The device is configured to block and alert, and thus this test provides an indication the effectiveness of both the blocking and alert handling mechanisms. A continuous stream of exploits mixed with some legitimate sessions is transmitted through the device at a maximum of 100Mbps (max 50,000 packets per second, average packet sizes in the range of 120-350 bytes) for 8 hours with no additional background traffic. This is not intended as a stress test in terms of traffic load - merely a reliability test in terms of consistency of blocking performance.*

  The device is expected to remain operational and stable throughout this test, and to block 100 per cent of recognisable exploits, raising an alert for each. Results are presented as a simple PASS/FAIL. If any recognisable exploits are passed - caused by either the volume of traffic ***or*** *the sensor failing open for any reason - this will result in a FAIL.*

- ***Test 6.1.2 - Passing Legitimate Traffic Under Extended Attack:*** *This test is identical to 6.1.1, where we expose the external interface of the device to a constant stream of alerts over an extended period of time. The device is expected to remain operational and stable throughout this test, and to pass 100 per cent of legitimate traffic. Results are presented as a simple PASS/FAIL. If any legitimate traffic is blocked - caused by either the volume of traffic* ***or*** *the sensor failing closed for any reason - this will result in a FAIL.*

- ***Test 6.1.3 - ISIC/ESIC/TCPSIC/UDPSIC/ICMPSIC:*** *This test attempts to stress the protocol stack of the device under test by exposing it to traffic from the ISIC test tool. The ISIC test tool host is connected directly to the external interface of the sensor, and the ISIC target directly to the internal interface. ISIC traffic is transmitted through the sensor (without passing through any other network equipment) and the effects noted. Traffic load is a maximum of 350Mbps and 60,000 packets per second (average packet size is 690 bytes). Results are presented as a simple PASS/FAIL - the device is expected to remain operational and capable of detecting and blocking exploits throughout the test to attain a PASS.*

Section 7 – Management and Configuration

The aim of this section is to determine the features of the management system, together with the ability of the management port on the device under test to resist attack.

Test 7.1 - Management Port

Clearly the ability to manage the alert data collected by the sensor is a critical part of any IDS/IPS system. For this reason, an attacker could decide that it is more effective to attack the management interface of the device than the detection interface.

Given access to the management network, this interface is often more visible and more easily subverted than the detection interface, and with the management interface disabled, the administrator has no means of knowing his network is under attack.

- ***Test 7.1.1 - Open ports:*** *We will scan the open ports and active services on the management interface and report on known vulnerabilities.*
- ***Test 7.1.2 - ISIC/ESIC/TCPSIC/UDPSIC/ICMPSIC:*** *This test attempts to stress the protocol stack of the management interface of the device under test by exposing it to traffic from the ISIC test tool. The ISIC test tool host is connected directly to the management interface of the IPS sensor, and that interface is also the target. ISIC traffic is transmitted to the management interface of the IPS device (without passing through any other network equipment) and the effects noted.*

  Traffic load is a maximum of 350Mbps and 60,000 packets per second (average packet size is 690 bytes). Results are presented as a simple PASS/FAIL - the device is expected to remain (a) operational and capable of detecting and blocking exploits, and (b) capable of communicating in both directions with the management server/console throughout the test to attain a PASS.

  Test 7.1.3 - *We note whether the ISIC attacks themselves are detected by the sensor even though targeted at the management port.*

NFR Sentivist Smart Sensor 100C V1.3 Test Results

Section 1 - Detection Engine

Test 1.1 – Attack Recognition	Attacks	Default ARRD	Default ARRB	Custom ARRD	Custom ARRB
Test 1.1.1 - Backdoors	7	6	6	7	7
Test 1.1.2 - WINS/DNS	3	2	2	3	3
Test 1.1.3 - DOS	10	9	10	9	10
Test 1.1.4 - False negatives (modified exploits)	14	9	9	11	11
Test 1.1.5 - Finger	4	4	4	4	4
Test 1.1.6 - FTP	5	5	5	5	5
Test 1.1.7 - HTTP	43	37	37	40	40
Test 1.1.8 - ICMP	2	2	2	2	2
Test 1.1.9 - Reconnaissance	8	7	7	8	7
Test 1.1.10 - RPC	9	6	6	8	8
Test 1.1.11 - SSH	1	1	1	1	1
Test 1.1.12 - Telnet	1	1	1	1	1
Test 1.1.13 - Database	1	1	1	1	1
Test 1.1.14 - Mail	1	1	1	1	1
Test 1.1.15 - Voice	1	0	0	0	0
Total	**110**	**91 / 110**	**92 / 110**	**101 / 110**	**101 / 110**
		83%	**84%**	**92%**	**92%**

Test 1.2 – Resistance to False Positives	Default	Custom
Test 1.2.1 - Suspicious FTP traffic	PASS	PASS
Test 1.2.2 - HTTP “exploit” using incorrect method	FAIL	PASS
Test 1.2.3 - Retrieval of Web page containing “suspicious” URLs	PASS	PASS
Test 1.2.4 - Simple SMTP QUIT command	PASS	PASS
Test 1.2.5 - Normal NetBIOS copy of “suspicious” files	PASS	PASS
Test 1.2.6 - Normal NetBIOS traffic	PASS	PASS
Test 1.2.7 - POP3 e-mail containing “suspicious” URLs	PASS	PASS
Test 1.2.8 - POP3 e-mail with “suspicious” DLL attachment	PASS	PASS
Test 1.2.9 - POP3 e-mail with “suspicious” Web page attachment	PASS	PASS
Test 1.2.10 - SMTP e-mail transfer containing “suspicious” URLs	PASS	PASS
Test 1.2.11 - SMTP e-mail transfer with “suspicious” DLL attachment	PASS	PASS
Test 1.2.12 - SMTP e-mail transfer with “suspicious” Web page attachment	PASS	PASS
Test 1.2.13 - SNMP V3 packet with invalid parameter	PASS	PASS
Test 1.2.14 - Fake DNS /bin/sh buffer overflow	PASS	PASS
Test 1.2.15 - Inter-firewall communication traffic	FAIL	FAIL
Test 1.2.16 - Fake SQL Slammer traffic	PASS	PASS
Test 1.2.17 - File copy of GIF file (contains bytes which look like NOP sled)	PASS	PASS
Total Passed	**15 / 17**	**16 / 17**

Section 2 - IPS Evasion

Test 2.1 – Evasion Baselines	Detected?	Blocked?
Test 2.1.1 - NSS Back Orifice ping	YES	YES
Test 2.1.2 - Back Orifice connection	YES	YES
Test 2.1.3 - FTP CWD root	YES	YES
Test 2.1.4 - ISAPI printer overflow	YES	YES
Test 2.1.5 - Showmount export lists	YES	YES
Test 2.1.6 - Test CGI probe (/cgi-bin/test-cgi)	YES	YES
Test 2.1.7 - PHF remote command execution	YES	YES
Total	**7 / 7**	**7 / 7**

Test 2.2 – Packet Fragmentation/Stream Segmentation	Detected?	Decoded?	Blocked?
Test 2.2.1 - IP fragmentation - ordered 8 byte fragments	YES	YES	YES
Test 2.2.2 - IP fragmentation - ordered 24 byte fragments	YES	YES	YES
Test 2.2.3 - IP fragmentation - out of order 8 byte fragments	YES	YES	YES
Test 2.2.4 - IP fragmentation - ordered 8 byte fragments, duplicate last packet	YES	YES	YES
Test 2.2.5 - IP fragmentation - out of order 8 byte fragments, duplicate last packet	YES	YES	YES
Test 2.2.6 - IP fragmentation - ordered 8 byte fragments, reorder fragments in reverse	YES	YES	YES
Test 2.2.7 - IP fragmentation - ordered 16 byte fragments, fragment overlap (favour new)	YES	YES	YES
Test 2.2.8 - IP fragmentation - ordered 16 byte fragments, fragment overlap (favour old)	YES	YES	YES
Test 2.2.9 - TCP segmentation - ordered 1 byte segments, interleaved duplicate segments with invalid TCP checksums	YES	YES	YES
Test 2.2.10 - TCP segmentation - ordered 1 byte segments, interleaved duplicate segments with null TCP control flags	YES	YES	YES
Test 2.2.11 - TCP segmentation - ordered 1 byte segments, interleaved duplicate segments with requests to resync sequence nos. mid-stream	YES	YES	YES
Test 2.2.12 - TCP segmentation - ordered 1 byte segments, duplicate last packet	YES	YES	YES
Test 2.2.13 - TCP segmentation - ordered 2 byte segments, segment overlap (favour new)	YES	YES	YES
Test 2.2.14 - TCP segmentation - ordered 1 byte segments, interleaved duplicate segments with out-of-window sequence numbers	YES	YES	YES
Test 2.2.15 - TCP segmentation - out of order 1 byte segments	YES	YES	YES
Test 2.2.16 - TCP segmentation - out of order 1 byte segments, interleaved duplicate segments with faked retransmits	YES	YES	YES
Test 2.2.17 - TCP segmentation - ordered 1 byte segments, segment overlap (favour new)	YES	YES	YES
Test 2.2.18 - TCP segmentation - out of order 1 byte segments, PAWS elimination (interleaved dup segments with older TCP timestamp options)	YES	YES	YES
Test 2.2.19 - IP fragmentation - out of order 8 byte fragments, interleaved duplicate packets scheduled for later delivery	YES	YES	YES
Test 2.2.20 - TCP segmentation - ordered 16 byte segments, segment overlap (favour new (Unix))	YES	YES	YES
Total	**20 / 20**	**20 / 20**	**20 / 20**

Test 2.3 – URL Obfuscation	Detected?	Decoded?	Blocked?
Test 2.3.1 - URL encoding	YES	YES	YES
Test 2.3.2 - /./ directory insertion	YES	YES	YES
Test 2.3.3 - Premature URL ending	YES	YES	YES
Test 2.3.4 - Long URL	YES	YES	YES
Test 2.3.5 - Fake parameter	YES	YES	YES
Test 2.3.6 - TAB separation	YES	YES	YES
Test 2.3.7 - Case sensitivity	YES	YES	YES
Test 2.3.8 - Windows \ delimiter	YES	YES	YES
Test 2.3.9 - Session splicing	YES	YES	YES
Total	**9 / 9**	**9 / 9**	**9 / 9**

Test 2.4 – Miscellaneous Obfuscation Techniques	Detected?	Decoded?	Blocked?
Test 2.4.1 - Altering default ports	NO[1]	NO[1]	NO[1]
Test 2.4.2 - Inserting spaces in FTP command lines	NO	NO	NO
Test 2.4.3 - Inserting non-text Telnet opcodes in FTP data stream	NO	NO	NO
Test 2.4.4 - Polymorphic mutation (ADMmutate)	YES	YES	YES
Test 2.4.5 - Altering protocol and RPC PROC numbers	YES	YES	YES
Test 2.4.6 - RPC record fragging (MS-RPC and Sun)	YES	YES	YES
Test 2.4.7 - HTTP exploits to port <> 80	YES	YES	YES
Total	**4 / 7**	**4 / 7**	**4 / 7**

Section 3 - Stateful Operation

Test 3.1 – Stateless Attack Replay	Alert?	Blocked?	Pass/Fail
Test 3.1.1 - Stateless Web exploits	NO[2]	NO[2]	PASS
Test 3.1.2 - Stateless FTP exploits	NO[2]	NO[2]	PASS

Test 3.2 – Simultaneous Open Connections (default settings)							
Number of open connections	10,000	25,000	50,000	100,000	250,000	500,000	1,000,000
Test 3.2.1 - Attack Detection	PASS	PASS	PASS	PASS	PASS	PASS	PASS
Test 3.2.2 - Attack Blocking	PASS	PASS	PASS	PASS	PASS	PASS	PASS
Test 3.2.3 - State Preservation	PASS	PASS	PASS	PASS	PASS	PASS	PASS
Test 3.2.4 - Legitimate traffic blocking	PASS	PASS	PASS	PASS	FAIL	FAIL	FAIL

Test 3.3 – Simultaneous Open Connections (after tuning)							
Number of open connections	10,000	25,000	50,000	100,000	250,000	500,000	1,000,000
Test 3.3.1 - Attack Detection	PASS	PASS	PASS	PASS	PASS	PASS	PASS
Test 3.3.2 - Attack Blocking	PASS	PASS	PASS	PASS	PASS	PASS	PASS
Test 3.3.3 - State Preservation	PASS	PASS	PASS	PASS	PASS	PASS	PASS
Test 3.3.4 - Legitimate traffic blocking	PASS	PASS	PASS	PASS	FAIL	FAIL	FAIL

Section 4 - Detection/Blocking Performance Under Load

Test 4.1 – UDP traffic to random valid ports		25Mbps	50Mbps	75Mbps	100Mbps	Max
Test 4.1.1 - 256 byte packet test - max 45,500pps	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	
Test 4.1.2 - 550 byte packet test - max 22,000pps	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	
Test 4.1.3 - 1514 byte packet test - max 12,000pps	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	

Test 4.2 – HTTP "maximum stress" traffic with no transaction delays		25Mbps	50Mbps	75Mbps	100Mbps	Max
Test 4.2.1 - Max 250 connections per second - ave packet size 1000 bytes - max 12,000 packets per second	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	
Test 4.2.2 - Max 500 connections per second - ave packet size 540 bytes - max 22,500 packets per second	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	
Test 4.2.3 - Max 1000 connections per second - ave packet size 440 bytes - max 27,500 packets per second	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	
Test 4.2.4 - Max 2000 connections per second - ave packet size 360 bytes - max 32,000 packets per second	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	

Test 4.3 – HTTP "maximum stress" traffic with transaction delays		25Mbps	50Mbps	75Mbps	100bps	Max
Test 4.3.1 - Max 500 connections per second - ave packet size 540 bytes - max 22,500 packets per second - 10 sec delay - max 5,000 open connections	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	
Test 4.3.2 - Max 1000 connections per second - ave packet size 440 bytes - max 27,500 packets per second - 10 sec delay - max 10,000 open connections	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	

Test 4.4 – Protocol mix		25Mbps	50Mbps	75Mbps	100Mbps	Max
Test 4.4.1 - 72% HTTP (540 byte packets) + 20% FTP + 6% UDP (256 byte packets). Max 400 connections per second - ave packet size 540 bytes - max 22,000 packets per second - max 75 open connections	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	

Test 4.5 – Real World traffic		25Mbps	50Mbps	75Mbps	100Mbps	Max
Test 4.5.1 - Pure HTTP (simulated browsing session on NSS Web site). Max 475 connections per second - 4 new users per second - ave packet size 560 bytes - max 21,000 packets per second	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	
Test 4.5.2 - Protocol mix - 72% HTTP (simulated browsing sessions as 2.5.1) + 20% FTP + 6% UDP (256 byte packets). Max 375 connections per second - ave packet size 560 bytes - max 21,000 packets per second - max 150 open connections	Detected	100%	100%	100%	100%	100Mbps
	Blocked	100%	100%	100%	100%	

Section 5 - Latency & User Response Times

Test 5.1 – Latency	Packet Size	25Mbps	50Mbps	75Mbps	100Mbps
Test 5.1.1 Average latency (µs) with no background traffic	256	136.09	150.91	158.27	178.16
	550	155.38	156.30	159.53	180.78
	1000	195.78	196.10	197.78	198.10
Test 5.1.2 Average latency (µs) with background traffic (50Mbps HTTP traffic, max 250 connections per second - ave packet size 540 bytes - max 12,000 packets per second)	256	160.46			
	550	178.21			
	1000	217.08			
Test 5.1.3 Average latency (µs) when under attack (10Mbps SYN flood (14800pps))	256	8061.87			
	550	6728.22			
	1000	6267.88			

Test 5.2 – User Response Times	Attempted Trans	Failed Trans	Min Page Response	Max Page Response	Ave Page Response
Test 5.2.1 - Web page response (ms) with no background traffic (50Mbps HTTP traffic, max 250 connections per sec - ave packet size 540 bytes - max 12,000 packets per sec)	172632	0	201	55398	246
Test 5.2.2 - Web page response (ms) when under attack (10Mbps HTTP traffic, max 250 connections per sec - ave packet size 540 bytes - max 12,000 packets per sec PLUS 10Mbps SYN flood (14800pps))	171527	139672	201	206780	45682

Section 6 - Stability & Reliability

Test ID	Result
Test 6.1.1 - Blocking Under Extended Attack	100%
Test 6.1.2 - Passing legitimate traffic under extended attack	100%
Test 6.1.3 - ISIC/ESIC/TCPSIC/UDPSIC/ICMPSIC	PASS

Section 7 - Management Interface

Test ID	Result
Test 7.1.1 - Open Ports	PASS
Test 7.1.2 - ISIC/ESIC/TCPSIC/UDPSIC/ICMPSIC	PASS
Test 7.1.3 - ISIC attacks detected against management interface?	NO

Notes:

1. Depends on backdoor package. Detection on multiple ports for BO2K package was disabled for performance reasons.
2. Requires configuration. By default, Smart Sensor 100C alerts on stateless exploits and will block the packets. Can be configured to suppress alerts and pass packets. Cannot be configured to suppress alerts and block packets.

Section 1: Detection Engine

We installed one sensor with the latest Package updates, and enabled all attack/exploit signatures (recorders, audit and policy enforcement signatures were disabled). NFR recommends the following critical adjustments in high performance environments, which were subsequently implemented for this test:

- *Disable NetbusPro package*
- *Disable Remotenc package*
- *Disable SubSeven package*
- *Disable BO2K detection across all ports (single port only)*
- *Disable server response monitoring (effectively disables all server-to-client signatures)*

Out of the box, signature recognition was good at 83 per cent, and was improved to 92 per cent following the application of a signature update after 48 hours. Blocking performance was identical following the update.

Performance in our “false negative” tests was adequate out of the box, and improved slightly following the signature update. However, there were still three misses out of the 14 test cases following the signature update, which was unusual given that NFR has a good track record for writing signatures for vulnerabilities rather than exploits. The product also did very well at detecting most of the variants that we include alongside our basic test cases (i.e. if five different exploits are known for a vulnerability, the signature is generally written well enough to detect all of them).

A major concern in deploying an IPS is the blocking of legitimate traffic. The device failed two test cases in our false positive test suite, and this was reduced to one (fake FireWall-1 RDP traffic) following the signature update. However, no other false positives were noted during stress testing with either Avalanche traffic or real-world traffic mixes.

Section 2: IPS Evasion

Resistance to known evasion techniques was excellent, with the Smart Sensor 100C achieving a clean sweep across the board in most of our evasion tests. *Fragroute* and *Whisker* both failed to deceive the device into ignoring valid attacks, and all of the attempts were decoded accurately.

Of the miscellaneous evasion techniques, changing ports on backdoors, inserting spaces into FTP commands and non-text Telnet opcodes in FTP data streams all proved troublesome, but the rest (including extensive RPC record fragging) were handled well.

Section 3: Stateful Operation

Out of the box, NFR claims that the Smart Sensor 100C can handle up to 500,000 open connections, depending on the type of traffic passing through the device at any point in time. The Smart Sensor 100C architecture incorporates what the engineers call a “*pressure system*”, meaning that instead of relying on fixed parameters to determine how much memory is allocated to critical resources such as state tables, the device dynamically allocates memory as needed.

This memory can be released should it be required by other subsystems as the system comes under different types of load. No tuning is possible, and the effects are not entirely deterministic from test to test. Results will undoubtedly be different in a real-world deployment.

During testing, we verified the Smart Sensor 100C's ability to handle up to 140,000 simultaneously open connections whilst successfully maintaining state on our half-open exploits. Occasionally, however, we noted a much lower maximum, depending on the behaviour of the pressure system in response to different traffic mixes.

During testing, the default operation of the device was to reject new connections when the state tables were full or resources were low, and this meant that once we exceeded the connection limit the device continued to maintain state on existing connections (thus detecting our half-open exploits), but inevitably, as a consequence, began to block legitimate traffic. This behaviour is not configurable.

Unusually, stateless "exploits" are alerted upon by default, and blocked. We do not believe that this is correct behaviour, since it leaves the device vulnerable to *Stick* and *Snot* tools (although the traffic would be blocked, of course). It is possible to configure the device to suppress alerts and pass the mid-flow traffic, but our preferred configuration - suppress alerts and block mid-flows - was not possible (NFR state that it could be achieved via the use of additional N-Code and a filter, however).

Section 4: Detection/Blocking Performance Under Load

Note that the NFR Sentivist Smart Sensor 100C was tested as a 100Mbps IPS device.

Performance at all levels of our load tests was impeccable, with 100 per cent of all attacks being detected and blocked under all load conditions.

We would happily confirm NFR's 100Mbps rating for this device, and would deem it conservative in most normal networks.

Section 5: Latency & User Response Times

The basic latency figures of the NFR Smart Sensor 100C were excellent for a 100Mbps device across the board under all traffic loads. They ranged from 136µs with 25Mbps of 256 byte packets, to 198µs with 100Mbps of 1000 byte packets.

Behaviour throughout the tests with no background traffic was consistent and predictable, with small increases as additional network load was applied from 25Mbps to 100Mbps. Placing the device under a half load of 50Mbps of HTTP traffic increased latency slightly, rising from 136µs to 160µs with 256 byte packets, from 155µs to 217µs with 550 byte packets, and from 195µs to 217µs with 1000 byte packets.

Given such low latencies, HTTP response times were unusually high with an average of 246ms with 50Mbps of HTTP traffic. On examining the results closely, we noted that much of the traffic during the test had latency closer to the norm of 201ms, but that there were periods of extremely poor response (55 seconds!) which adversely affected the average.

This indicates that normal latencies are much more acceptable, but that frequent “housekeeping” by the sensor (perhaps as a result of the pressure system architecture) has a potentially adverse effect on performance through the device. This may not be as noticeable in real-world deployment, but is an issue that needs to be addressed by NFR.

The device also had significant problems with our DDOS test, and would perform best when situated behind an attack mitigation device or a firewall with DDOS mitigation capabilities. Latency when under 10Mbps (14800 packets per second) of SYN flood traffic was excessive, although UDP packet loss was minimal. The overall effect on HTTP traffic, however, was an increase in response times and a high number of failed transactions.

This is an issue which NFR recognises and is working on at the time of writing. It may be possible to configure the Smart Sensor to perform better in these situations by disabling all notice session-based filters (there were four of these enabled during the test) though we did not verify this and are not sure of the resulting effect on overall protection as a result.

Section 6: Stability & Reliability

NFR Sentivist Smart Sensor 100C performed consistently and completely reliably throughout our tests. Under eight hours of extended attack (comprising millions of exploits mixed with genuine traffic) it continued to block 100 per cent of attack traffic, whilst passing 100 per cent of legitimate traffic.

Exposing the sensor interface to ISIC-generated traffic had no long-term adverse effect on the device, although communications between the management console and the management server/sensor were intermittent during the attack. In addition, at the height of the attack the effect was similar to that noted when subjected to a SYN flood, causing some legitimate traffic to be blocked.

A large number of ISIC-related alerts were raised during the attack and the attack was mitigated partially (many malformed ISIC packets are also dropped silently). All other malicious traffic continued to be blocked successfully during the attack, and there were no residual stability problems once the ISIC attack had been terminated.

Section 7: Management Interface

Only a single port is open on the sensor - TCP/1968 - and this is not visible to port scanners.

The extended ISIC attack against the management interface caused sluggish communications between the console and the management server/sensor (due mainly to the extent of the ISIC flood), but these were restored successfully once the attack was terminated and there were no residual stability problems.