

Sentryware HIVE V3.0

Technical Evaluation

An NSS Group Report



First published October 2004 (Version 1.0)

Published by The NSS Group Ltd.
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>

©1991-2004 The NSS Group

All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the authors. This report shall be treated at all times as a confidential and proprietary report for internal use only.

Please note that access to or use of this Report is conditioned on the following:

1. The information in this Report is subject to change by The NSS Group without notice.
2. The information in this Report is believed by The NSS Group to be accurate and reliable, but is not guaranteed. All use of and reliance on this Report are at your sole risk. The NSS Group is not liable or responsible for any damages, losses or expenses arising from any error or omission in this Report.
3. NO WARRANTIES, EXPRESS OR IMPLIED ARE GIVEN BY THE NSS GROUP. ALL IMPLIED WARRANTIES, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE DISCLAIMED AND EXCLUDED BY THE NSS GROUP. IN NO EVENT SHALL THE NSS GROUP BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES, OR FOR ANY LOSS OF PROFIT, REVENUE, DATA, COMPUTER PROGRAMS, OR OTHER ASSETS, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
4. This Report does not constitute an endorsement, recommendation or guarantee of any of the products (hardware or software) tested or the hardware and software used in testing the products. The testing does not guarantee that there are no errors or defects in the products, or that the products will meet your expectations, requirements, needs or specifications, or that they will operate without interruption.
5. This Report does not imply any endorsement, sponsorship, affiliation or verification by or with any companies mentioned in this report.
6. All trademarks, service marks, and trade names used in this Report are the trademarks, service marks, and trade names of their respective owners, and no endorsement of, sponsorship of, affiliation with, or involvement in, any of the testing, this Report or The NSS Group is implied, nor should it be inferred.

TABLE OF CONTENTS

INTRODUCTION	1
Web Application Firewall	1
The NSS Web Application Firewall Test	2
SENTRYWARE HIVE V3.0	4
Executive Summary.....	4
Architecture.....	4
HIVE Appliance	4
Web Manager	7
Performance	8
Security Effectiveness	9
Usability	10
Installation.....	10
Configuration	11
Policy Management.....	14
Alert Handling	20
Reporting and Analysis.....	22
Verdict.....	22
Contact Details	23
APPENDIX A – TEST RESULTS.....	24
The Test Environment	24
Section 1 – Detection Engine	24
Section 2 – Evasion.....	26
Section 3 – Performance Under Load (No Security Policies Applied)	26
Section 4 – Performance Under Load (Security Policies Applied).....	28
Section 5 – Latency & User Response Times.....	29
Section 6 – Stability & Reliability	30
Section 7 – Management and Configuration	30
Sentryware HIVE V3.0 Test Results	32
Section 1 - Detection Engine	32
Section 2 - Evasion Techniques	32
Section 3 - Performance Under Load - No Security Policy	33
Section 4 - Performance Under Load - Security Policy Applied.....	33
Section 5 - Latency & User Response Times	34
Section 6 - Stability & Reliability	34
Section 7 - Management Interface	34

TABLE OF FIGURES

Figure 1 - HIVE: Configurations menu.....	12
Figure 2 - HIVE: Units menu.....	13
Figure 3 - HIVE: Domains.....	15
Figure 4 - HIVE: Global settings	16
Figure 5 - HIVE: Defining Start Pages	17
Figure 6 - HIVE: Defining HTAGs.....	18
Figure 7 - HIVE: Using HTAGs.....	19
Figure 8 - HIVE: Viewing log entries.....	20
Figure 9 - HIVE: Statistics summary screen	21

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, VPN's, 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.

INTRODUCTION

In a recent 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. Network-level (and some generic application-level) attacks can be prevented by the new breed of *Intrusion Prevention Systems* (IPS), but these are unable to detect or prevent subversion of application logic.

Web Application Firewall

The Internet has brought about a dramatic shift in the way business applications are deployed. In the past, our business applications were run on private servers by our own employees. Today, those business-critical applications are Web-based, running on public-facing servers and being used by external entities, such as customers and business partners.

This has created new and more complex security threats. Traditional approaches such as firewalls, Virtual Private Networks, Public Key Infrastructures and Intrusion Detection/Prevention Systems often cannot protect the application layer, which is the least secured and most vulnerable layer. These security products are designed to keep out or monitor the external entity, whereas the whole point of our Web applications is to welcome in external users and provide limited and controlled access to our corporate data.

Once the perimeter firewall has permitted the HTTP traffic as it has been instructed to do via its applied security policy, there remains a wealth of tactics to be employed by the malicious user in order to subvert the back-end application via the seemingly innocent HTTP protocol. It is thus possible for the Web hacker to target application-level vulnerabilities without running foul of perimeter firewalls - there will be no mangled, fragmented or oversized network packets, no mismatches between address and content. Instead, these application-level attacks employ subtle changes to otherwise valid commands, cookies that have been tampered with, or changes to hidden form fields.

Applications for the Web thus require a full range of security solutions designed to not only protect the host and network, but also the applications that run on them. The security measures must protect privileged information, whilst enabling the organisation to manage its environment and external users to access the application and its supporting data.

While the lower layers of the OSI model are well defined, the application layer provides a wide range of disparate network services. Although TCP/IP specifications are known worldwide, no two applications implement the same business logic and the same technology. Therefore no single "signature-based" approach can secure an application against its unique application-layer vulnerabilities. Applications typically require both read and write access to one or more databases, often with full privileges, and corporate databases can be sabotaged by inserting or concatenating various SQL commands to input fields or messages.

The *Web Application Firewall* thus works at the application layer – much higher than traditional solutions such as firewalls and IPS – to intercept all incoming and outgoing traffic to and from applications, validating and securing requests before they are allowed to pass through to back-end servers. These products understand the application logic, and have a detailed knowledge of the acceptable rules of engagement between the external client and the internal application server. They are thus capable of inspecting the content of each request and response and applying a complex set of rules in order to ensure that the client is not doing anything untoward.

The Web Application Firewall is also designed to regulate each application to prevent manipulation and defacement, providing a safe environment for corporate data.

The NSS Web Application Firewall Test

As part of its extensive Web Application Firewall test methodology (see section on *Testing Methodology* later in this report for details) The NSS Group subjects each product to a brutal battery of tests that verify the stability and performance of each device tested, determine the accuracy and effectiveness of its security coverage, and ensure that the device will not block legitimate traffic.

If a particular device 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 performance and security requirements, the NSS Group has developed a specialised lab environment that is able to exercise every facet of a Web Application Firewall product. The test suite contains hundreds of individual tests that evaluate these products in three main areas: *performance and reliability*, *security effectiveness*, 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.

SENTRYWARE HIVE V3.0

Executive Summary

HIVE is an HTTP firewall. It provides security for applications hosted on a content server, connected to a network via HTTP. These include Web applications, WAP applications, XML applications, and so on. HIVE reinforces the HTTP protocol as defined in the Internet standards, cancelling any attempt to defeat application security via incorrect requests or modifications to legitimate requests.

HIVE is provided as a 1U appliance with two copper Gigabit ports, and is positioned between the client and the server to intercept the HTTP traffic between them, verifying that the content is not malicious. The verification is based on information derived from the page contents - from a hidden field or form defined within the HTML code, for example - and no modifications to existing applications are necessary for HIVE to operate.

Out of the box, HIVE takes only minutes to install, though configuration can take longer depending on the complexity of the application(s) to be protected. Protection for simple Web sites can be implemented within a matter of minutes, however.

Performance is limited, as with all devices of this type, and the capacity of HIVE needs to be carefully matched to the capacity of the device(s) it is protecting and the Internet "pipe" to which it is attached.

Overall, the level of protection offered is excellent, especially give the often minimal amount of configuration required. We found the device to be very stable and reliable throughout our tests.

Architecture

The HIVE appliance-based Web Application Firewall consists of the following components:

HIVE Appliance

HIVE is based on the IBM x306 series server platform, and is thus provided as a 1U appliance with two copper Gigabit ports, which are used both for passing traffic (*internal* and *external* interfaces) **and** management.

Although only one in-line connection is supported, it is possible to place more than one Web server behind HIVE depending on processing requirements. It is equally possible to connect HIVE directly in-line with a single Web server for maximum performance. Given the way HIVE operates, it has no trouble interacting with load balancers if further scalability is required.

The device is positioned between the client and the server to intercept the HTTP traffic between them, and can operate either in transparent bridge mode or as a reverse proxy. HIVE filters all IP traffic, and ignores traffic destined for the servers that it is not configured to protect. It is also possible to configure HIVE to permit only HTTP traffic that is directed towards the domains it is configured to protect. In this mode HIVE, in addition to being a Layer Seven firewall, acts a firewall to block lower layer traffic.

For all domains it is configured to protect, HIVE intercepts the response from the content server and locates the input fields of the HTTP application (forms, fields for data entry, and so on) and signs them.

This is done by appending AES (*Advanced Encryption Standard*) cryptographic “tokens” to each modifiable element in the response of the content server. It is guaranteed that these tokens cannot be modified or decrypted before they are included into the content. The signed data is then transmitted to the client.

Sentryware calls the process of analysis, encryption, parsing and validation *Context Authentication (CA)*. Context Authentication is performed on the fly as the page is passed to the user by HIVE, and does not in any way affect the HTML content of the page, which looks exactly the same to the end user on the screen. Only by inspecting the source code of the signed page can the cryptographic tokens be viewed. This also means that when creating the page, the designer does not have to modify the content to accommodate HIVE.

HIVE does not use cookies to track users - all the information necessary is included in the page sent to the client. This means that no special configuration is needed for load balancers to function with HIVE and prevents the subversion of HIVE itself via cookie tampering.

The *Context Authentication* process consists of the following steps:

1. *Analyse the content of the page sent from the server to the client.*
2. *Abstract, from the point of view of the application, the important parts of the content and their location.*
3. *Generate one or more tokens which describes those values*
4. *Encrypt the tokens in a manner that prevents modification by the end user*
5. *Append the tokens to the elements of the Web page from which they were derived*
6. *Send the page to the client*

Once the client decides to submit a request which has been previously “signed” to the server, the request is intercepted by HIVE and the token is processed as follows:

1. *The token is decrypted by HIVE to obtain a plaintext version.*
2. *Once decrypted the token is compared with the arguments contained within the request, checking their nature, value, size and incidence.*
3. *If, and only if, the request agrees with all the parameters specified in the token, HIVE removes the token from the content and forwards the request to the server.*

HIVE validates input data based on the original information of the contents:

- ***Number of variables and names*** - *HIVE ensures the application receives exactly the number of variables expected, preventing access to any functionality unknown to the application, such as debug mode or back doors.*

- **Content values** - *If the values are implemented in the page (using hidden fields, configuration options, and so on) they must be identical in the client request. This prevents the user from modifying critical values, such as product codes and prices as a page is submitted.*
- **Value size** - *If the designer implements a size for the content, the data input will be validated accordingly. If this is not specified, input will be validated against a default maximum value configured within HIVE. This prevents attacks based on buffer or heap overflows*
- **Destination URL** - *A signed page permits the end user to access only a pre-determined number of URLs, and they are verified such that the user cannot jump directly to a URL without appropriate permission.*

This technology permits the authentication of the information to be included in each web page, thus removing the need to store such data externally (such as within the HIVE appliance), or to synchronise user information, authentication details, or session/state data between multiple nodes.

HIVE does more than protect the plain HTML. When an HTTP application establishes a cookie with the end user, HIVE intercepts the corresponding HTTP section and signs it. Thus it is possible to check the integrity of the cookie if it is sent once again to the server in future requests. This prevents a user from modifying the contents of the cookie, preventing any kind of cookie poisoning attack.

If any of the cryptographic tokens or cookies have been tampered with, or any of the items protected by those tokens have been tampered with (i.e. by amending a hidden price field from \$4999.00 to \$4.99) then HIVE blocks the request.

Should a user attempt to inject plainly malicious data - such as binary data or shell code - within a request, or execute a Cross Site Scripting or SQL injection attack, this is detected and blocked via optional rules-based protection mechanisms that can be enabled or disabled via a single check box for each.

Whenever a request is blocked, HIVE generates an error message in the log, with the option of sending a real-time alert to the administrator. The user will either see a HIVE-generated error message or will be redirected transparently to another page (depending on how the system has been configured).

Naturally there is a significant processing overhead to the Context Authentication procedure, but the biggest advantage of this method is that there is no requirement to make any changes to the back-end application - HIVE will work with almost any existing application right out of the box (some very poorly written applications may require minor modifications to allow HIVE to function correctly, but these will be few and far between).

At its most basic level, HIVE works on the concept of “*start pages*”, legal entry points in an application - the most secure application will have only one start page. From that point on, every page and object access will require a HIVE signature in order to be completed - thus, changing a URL or attempting to bypass the application’s logical flow by directly accessing another Web page (or an operating system resource such as `CMD.EXE` or `/etc/passwd`) will also generate an error condition.

Bear in mind that HTTP was defined as a document retrieval standard, not as an application language. Therefore the combination of requests and responses that make up an application cannot be defined in any sort of generic “signature”, in the same way we might define a signature against the Code Red exploit in a network-level IDS/IPS system. This makes Web application protection very much more difficult.

Sentryware likens this protection to a game of chess. The form of the game (the application behaviour) cannot be known in advance, and neither can your opponent’s (the client’s) behaviour. However, all the rules, playing conventions, pieces, board layout, and playing tendencies of the game of chess **are** known. The same is usually true of even the most complex Web application and the underlying protocols which support it.

HIVE is thus focused on enforcing the knowable rules. Only “legal” moves are permitted, and control over play is transparent, having little or no effect on the participants. HIVE strictly enforces HTTP rules (RFC 2616), performs analysis and interpretation of outbound responses, interprets and validates inbound data and requests based on the above, and tracks and interprets recent communication flow.

This not only allows it to protect the application as it stands today, but will also allow it to handle changes in the application transparently, unless the new sections of the application have their own set of logic rules which must be enforced. Certainly changes that do not alter the logic flow of the application - cosmetic changes such as page layout, and so on - will have no affect on HIVE’s operation. Ultimately, this also provides security for attacks that are not yet known.

Using multiple HIVE appliances in a load-balanced or fault-tolerant deployment is reasonably straightforward because there is no state information which needs to be maintained across devices. It is also worth noting that in reverse proxy mode the device can perform *Network Address Translation* (NAT) on traffic passing through it, translating from an external IP address and port to an internal IP address and port, thus facilitating the external publication of internal applications.

Web Manager

HTTPS access is provided to the HIVE appliance in order to use the browser-based configuration and management utility. The Web interface provides the means to configure all of the parameters used to manage HIVE, though not always in the most intuitive manner.

A simple two-tier architecture is employed, and policies are stored directly on the appliance - there is no separate management server available for HIVE (although HIVE is capable of logging to syslog or any SNMP platform). This means that by default, the Web GUI is intended to manage single devices, or small numbers of devices, and thus might not scale well in larger deployments.

Note that, unusually, there is no dedicated management port on the HIVE appliances. Instead, it is necessary to either manage over one of the active ports. We would prefer to see a dedicated management port on the device to ensure that congestion on the detection interfaces does not affect the management process.

Performance

The aim of this section is to verify that the device is capable of effectively supporting legitimate traffic, as well as detecting and blocking attacks, when subjected to increasing loads of traffic with varying packet sizes and HTTP response sizes. Web Application firewalls generally are far more limited in their performance than other network-level IPS systems, since they generally have a much heavier processing overhead. In addition, given their usual deployment position in front of a single Web server or small group of servers, their only real requirement is to be able to outperform the servers they protect in terms of connections per second and number of open connections supported.

UDP performance with a variety of packet sizes was poor with smaller packets (less than 512 bytes) and barely acceptable at larger packet sizes compared to a pure networking device. However, given the likely deployment scenario as mentioned above, its overall packet processing performance should be considered acceptable. The bottleneck with this type of device is always likely to be the CPU rather than the network performance.

With pure HTTP traffic, and no security policies applied, the device could handle a maximum of 2000 new connections per second. Whilst this is hardly blistering performance, it is adequate for the overall performance levels of the device once security policies are applied. Clearly, however, an administrator would need to consider this overall limit when deciding how many Web servers to place behind a single HIVE appliance.

HTTP performance with security policies enabled were felt to be within acceptable limits for a device of this type, ranging from 84cps to 500cps depending on the packet and HTTP response sizes. Note, however, that performance takes a further hit when HIVE is forced to sign objects on a page - maximum *transactions per second* falls from 260 to 24 for the equivalent HTTP response size, whilst *response time* increases 3 or 4-fold depending on network load.

In our "real world" tests, where genuine Web pages were mixed with associated GIF/JPG images to increase the overall response size, HIVE performs much better, achieving 192 transactions per second before response times become excessive and performance becomes erratic. Note that we found the device could cope with short bursts of traffic well in excess of these figures - the only effect being a temporary increase in HTTP response times.

Basic latency figures were just about within acceptable limits for a pure networking device with all packet sizes at traffic loads up to 250Mbps, ranging from 220 μ s with 250Mbps of 256 byte packets, to 303 μ s with 250Mbps of 1000 byte packets. With more typical (larger) packet sizes, latency remained acceptable up to 500Mbps (260 μ s with 550 byte packets and 318 μ s with 1000 byte packets), and it was around 800Mbps before the device began to drop packets and latency increased to unacceptable levels.

With layer 7 devices such as HIVE, microsecond latency is less of an issue than with network-level in-line devices such as Intrusion Prevention Systems.

Thus the latency figures seen here are well within acceptable limits for a device of this type. Of more relevance are the HTTP response times, which were acceptable across all tests up to 75-80 per cent of the maximum load (as determined in our tests).

Exposing the sensor interface to an extended run of ISIC-generated traffic had no adverse effect, and the device continued to detect and block all other exploits throughout and following the ISIC attack.

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

Security Effectiveness

We installed one sensor in-line between our “internal” and “external” networks. This device was initially configured to protect all our “internal” Web servers in the following way:

- *The URLs used by the Spirent Reflector were designated as “start pages” using a regular expression to ensure that each page was inspected by HIVE (these pages contain no objects for HIVE to sign)*
- *The home page of our NSS test Web site used during the Avalanche performance tests was designated as a “start page”. This page includes a total of 24 objects which must be signed by HIVE*
- *The home pages of our vulnerable Web sites were designated as “start pages” - further configuration was required during testing of the vulnerable Web servers to ensure all tests could be completed*
- *Cookie protection was enabled*
- *Shell code detection was enabled*
- *Buffer overflow protection was enabled*
- *Banner replacement was enabled (all banners replaced with “HIVE Inspected”)*

Most of our attempted attacks were blocked successfully with no further configuration. One of the most striking benefits of HIVE was that there was no need to perform any changes on any of our applications in order for it to function, and a high degree of protection was provided with a minimal amount of configuration of the product.

Additional configuration was required to enforce the client-side validation of our forms (to be expected), and to detect directory traversal and OS command concatenation attempts (which we would have expected to be covered by the default configuration). Buffer overflow protection, shell code detection, Cross Site Scripting protection and Cookie Poisoning prevention all worked well. We noted that Cross Site Scripting detection on POST requests concentrated on detecting the `<script>` tag in one of the fields posted - we would like to see that extended to check for other tags which can be used maliciously.

Interestingly although HIVE does not contain signatures or a detection engine for common HTTP vulnerabilities, it still managed to detect and block all of the exploits we attempted either via the ability to detect shell code, or via the ability to block direct access to disallowed pages (including CMD.EXE and other critical system files).

Overall, we were very impressed with the high level of protection provided for such a minimum amount of configuration.

Resistance to HTTP evasion/obfuscation techniques was excellent, with HIVE achieving a clean sweep across the board in our evasion tests. Every technique was detected and blocked effectively, and only one was not successfully decoded.

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

When installing HIVE for the first time in the network, it is necessary to configure a series of parameters for the product to function correctly. This configuration is carried out using the serial port of the HIVE, connected to the PC of the administrator via a null-modem cable. The console can also be accessed directly by connecting a monitor and keyboard to HIVE.

Initial installation of the appliance consists of little more than connecting the network cables and powering on the device, at which point a simple text-based wizard steps the administrator through the basic device-level configuration.

The first time this is run, it prompts for an administrator user name and password before entering the *Rapid Configuration* mode, where the basic configuration parameters are entered. These include IP address, subnet mask, the TCP port for the web interface, routes and DNS servers. When the Rapid Configuration has been completed, a menu appears from which it is possible to verify or modify these same parameters, as well as upgrade the firmware, set date and time, re-load the default system configuration file and view the system status screen. The information available on the Status screen includes:

- **HIVE Version** - *The current version of firmware installed.*
- **System Date** - *Shows the system date and time.*
- **CPU Usage** - *Shows current CPU utilisation.*
- **Memory Usage** - *Shows current memory used by HIVE*
- **Ethernet One/Two** - *Shows received and transmitted data information for both Ethernet interfaces.*

HIVE has an IP address purely to provide access to the web interface from the network. However, functioning as a bridge, HIVE acts transparently and forwards packets without any type of modification, encapsulation or translation, and thus IP addresses are not applied to the main interfaces. We would prefer to see a separate dedicated management port, however, rather than use the detection ports.

For additional security, it is possible to control access to the Web configuration interface. This is done by filtering the IP addresses that are permitted to access the management IP address, thus preventing unauthorised connections.

The permitted IP addresses can be defined in a list only from the Web interface, but the ACL feature can be activated or deactivated from the administrative console.

Although the HIVE software arrives pre-installed on the appliance, we did attempt to re-load the software during the test. After backing up the configuration, we inserted the CD-ROM and rebooted the appliance. The automated install commenced (following confirmation from us) and once it had completed we re-loaded the configuration file - the appliance was ready to go in less than ten minutes from the point of power cycling the device. Very impressive.

Documentation is restricted to just two manuals - an *Installation Guide* and an *Administrator's Guide*. They are provided in both electronic (PDF) and hard copy versions, and are amongst the best documentation we have seen in our labs.

The *Installation Guide* is very extensive for a device that is so simple to install. It goes into great detail on general networking, IP addressing and routing concepts and we think this is to be commended. Whilst it is clear that no one who does not already have a clear grasp of these concepts should be installing and configuring a device such as this, that is - sadly - not always the case. It is nice to see an installation guide hand-hold the administrator from the very basic steps through to getting the product up and running.

Once it is running, the Administrator's Guide will provide all the information required to configure and maintain it. It provides much more than the usual "click here, this happens" type of "instruction" that we have come to expect, instead providing a wealth of information on why a particular parameter has been provided and what happens when it is used. The only thing missing was a reference section on the use of regular expressions.

Overall, the level of detail was excellent, and we found coverage of all the main features to be extremely comprehensive and very clear. Top marks for documentation.

Configuration

Configuration of the HIVE appliance is performed via the Web interface over a secure HTTPS connection. The administrator logs in using either the main admin ID and password, or a lower-level user name and password, and it is possible to log in to only one HIVE device at a time. All configuration information is stored directly on the device being managed - there is no "middle tier" centralised management server involved. This can work well with only a small number of devices to manage, but will not scale well in larger deployments.

Each "configuration" (akin to a security policy for a particular application) within HIVE can have a unique user name associated with it, and that user will only be presented with his own configuration (and none of the administrative functions) when logging in. However, if one user requires access to several different configurations (perhaps in a managed service environment) he would require several different login identities - one for each.

We would like to see this extended to more of a role-based user access model, with different levels of access within each configuration (read only, log access only, full configuration rights, etc.), as well as the ability to define users independently of configurations and assign multiple configurations to each user.

Four main menu entries along the top of the screen provide access to the *Administration*, *Units*, *Configuration* and *Logs* sections of HIVE.

The *Administration* menu contains high-level options for enabling/disabling the ACL list (restricting access to the management interface to specified hosts), importing and managing digital certificates and certificate requests (for SSL support), managing licenses, backing up and restoring the global configuration file for the device, changing the admin user name and password, and viewing some basic HIVE statistics (number of packets processed, number of legitimate requests, number of attacks blocked, and so on). This menu option is available only to the main administrator.

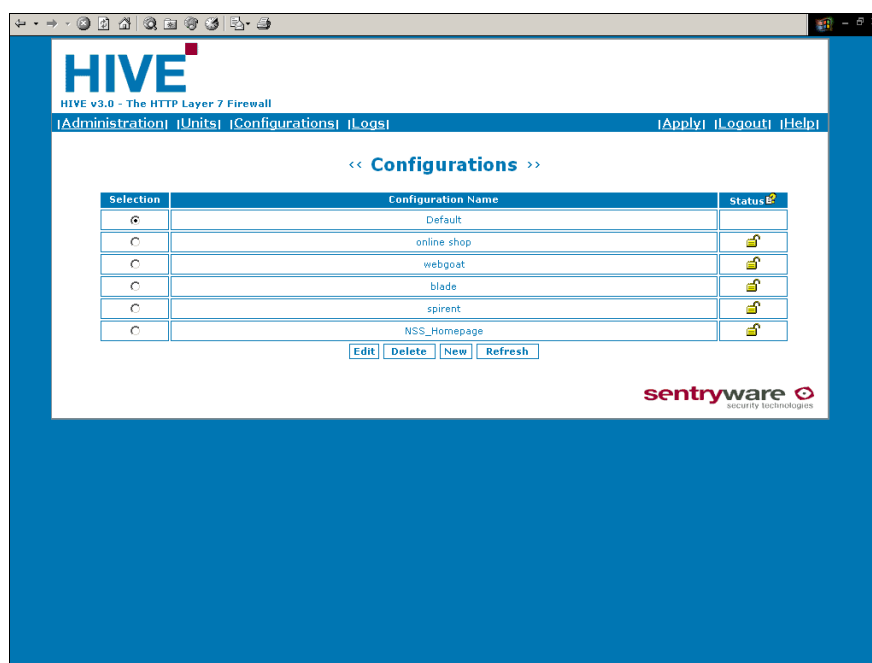


Figure 1 - HIVE: Configurations menu

The *Units* menu is to configure settings for individual HIVE appliances. The IP address, DNS servers and routing information that were created during the initial appliance installation are mirrored - and can be further modified - here.

The *NAT* option allows the creation of a *Network Address Translation* table, which is used to publish internal servers external IP addresses. This takes requests made to a public IP address and maps them to a different IP address and port on the internal system.

For those who do not wish to rely solely on checking the HIVE logs via the console, it is possible to use the *Alerts* menu to configure external syslog and SMTP servers (the latter to enable transmission of alert e-mails).

HIVE is designed to handle the following types of attack:

- **Vulnerable test applications** - Default installations of Web server software often include test pages and applications designed to demonstrate the capabilities of the server to a new user. These applications are sometimes vulnerable to attacks and are actively exploited by crackers. HIVE prevents access to pages that are not directly referenced by the Web site, such as test applications, files unintended for publication, database files, the registry files of the Web site, private documents, and so on.
- **Implementation problems with the content server** - Web servers can have problems with implementation, such as the UNICODE programming error discovered in IIS, or the SHTML memory overflow in I-Planet 4.0. Both vulnerabilities are protected by HIVE by restricting access to pages not part of the application and preventing the use of shell code.
- **Cookie manipulation** - Applications often use cookies with the aim of maintaining session information for a user or to store temporary information such as names or access keys. The modification of these values can create serious security problems. HIVE prevents this by allowing cookie content to be signed.

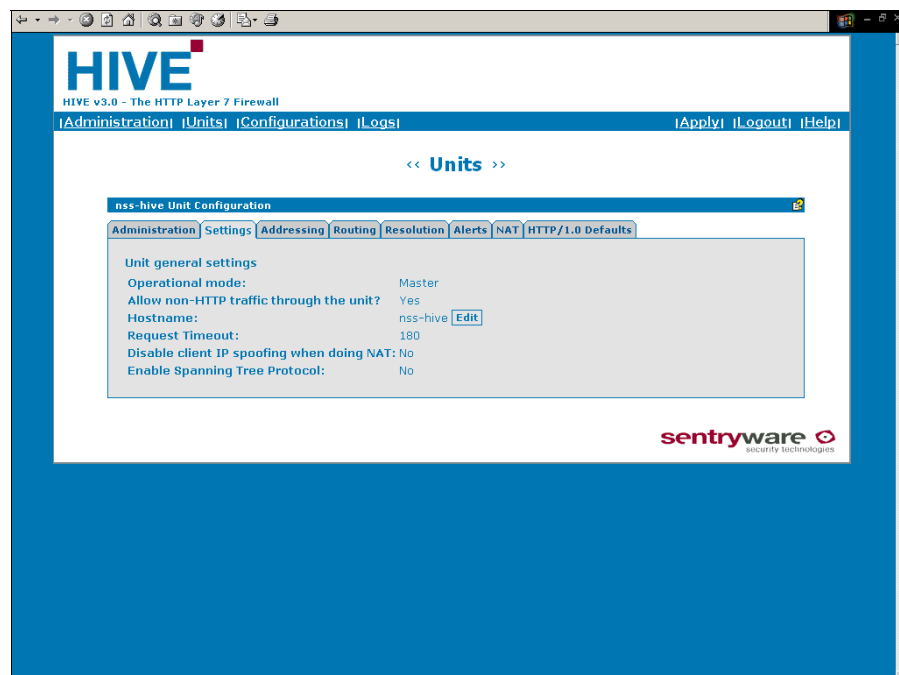


Figure 2 - HIVE: Units menu

- **Validation of data input** - Applications should validate all inputs made by a client. For example, an application which has an input field that accepts an e-mail address can be attacked by entering commands that can be executed on the content server instead of the required address. The application should filter illegal characters (that is to say accept the entered data, transform it and then send the transformed data) by identifying the entry point of each one and then verifying the client data. HIVE performs this task by allowing a programmer to insert the validation information in the appropriate values in the Web page and automatically verifying and enforcing these values when the request is received from the client.

- **Hidden field tampering** - Applications can often store information about a session or other data (such as product codes or prices) in "hidden fields". These fields are not displayed to the user, but are accessible within the HTML source code of the page, or within the URL in the browser, and can thus easily be modified by a malicious user. HIVE protects these fields using content signatures, which, if the field is modified register an alert and block the attack.
- **Buffer overflows** - The HTML source code of a page can force a browser to allow only a limited number of characters within an input field. This limitation is imposed on the client but can easily be bypassed. The applications that trust these limits in content length can often be faced with problems buffer and heap overflows, which can ultimately permit remote access to the server. HIVE detects and reinforces the maximum length defined for a field and registers and blocks attacks of this kind.
- **Backdoors** - By means of entering specific parameters in an application, it could be possible to access a hidden backdoor or a debug tool that, if activated, could display important information. It could also permit a client to control the server remotely. HIVE renders this impossible, by blocking requests to resources other than defined applications.
- **Cross-Site Scripting (CSS/XSS)** - It is possible to "trick" a browser into running malicious scripting code when a user clicks on a link, or a user can submit malicious code in a form to be uploaded - such as a guest book application.
- **SQL Injection** - Back-end databases contain a wealth of information about the application and its users. At the very least, they will contain lists of user names and passwords, along with customer data. At worst (from the point of view of being compromised) they might contain credit card numbers and other more personal data. HIVE prevents the user from using SQL commands in input fields to extract this information.

Policy Management

There is only a single policy on each device, and this takes the form of the global HIVE configuration file. This file can be backed up to a local or network hard drive remote from the appliance, and can later be reloaded to the appliance to restore a previous HIVE configuration.

The *Configuration* menu provides access to the **application** Configurations. Each Web application can have its own Configuration, and multiple application Configurations are stored within the global HIVE configuration. This overlap in terminology is confusing - we will refer to the application Configurations with a capital "C" throughout this report. It is not possible to save and load individual application Configurations - the entire HIVE configuration for a device is saved in a single file. This file also contains the network addressing information for an appliance, so it is not possible to create a single policy for distribution to multiple appliances (at least not without some manual editing of the XML file).

When the system is first installed, only the *Default Configuration* exists. This defines various items that are copied automatically each time a new Configuration is created (such as whether cookie poisoning protection, shell code protection and SQL injection detection are enabled), or whenever the default values are loaded.

The Default Configuration can therefore be thought of as a global “template” and can not be deleted.

Each new Configuration created defines all the features of a specific Web application required for HIVE to protect it, and each is assigned a unique user name and password. When a particular user logs in, that user will only be able to access his own Configuration and will only be able to see the log entries raised via that Configuration. As mentioned before, the user name is superfluous in anything but a very basic service provider environment. If there is only one administrator to the system, the requirement to allocate a separate and unique user for every Configuration is annoying - in most environments where multiple administrators are required, it would be more useful to be able to assign multiple Configurations to each user.

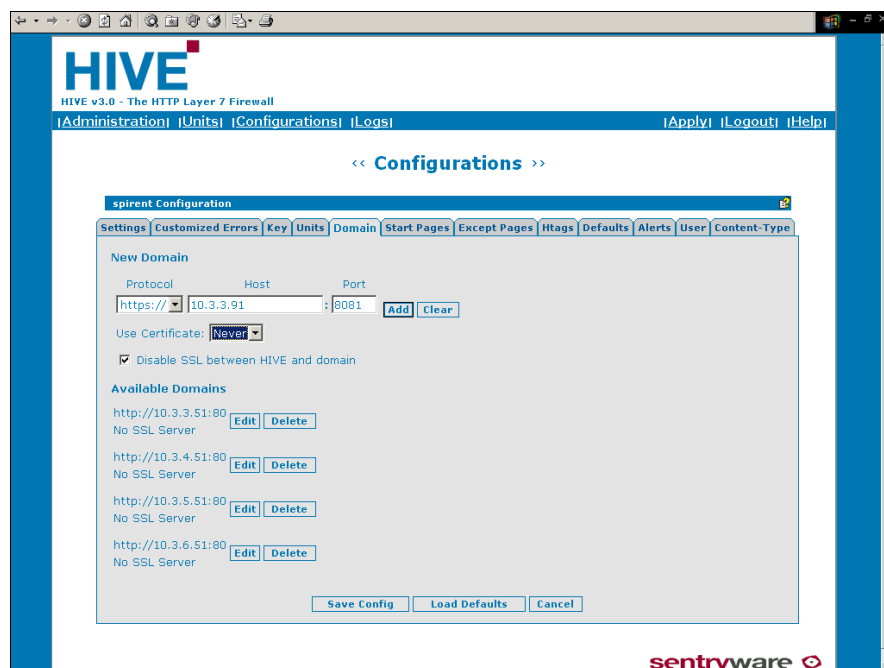


Figure 3 - HIVE: Domains

Creating new Configurations is far from intuitive. There is a set sequence of events that must be completed in order to finalise a Configuration, and the only way to step through this sequence at present is to attempt to save the Configuration at various points and allow the subsequent error message to inform you which steps have not yet been completed. This process is crying out for a *Wizard* feature that will take the administrator through the procedure in the required order.

One feature we liked here was the *EZ HIVE* option. This provides an almost “*plug and play*” element to the otherwise complex HIVE configuration that allows an inexperienced user to enjoy a high level of web security without the need for programming more options.

EZ HIVE is ideal for protecting multiple domains with HIVE, for example in a hosting (or ISP) environment, or where it is more important to provide broad coverage in the quickest amount of time rather than go for the highest levels of security.

The main features of the EZ HIVE mode are :

- *HIVE only signs objects that access the application - for example forms, or links that include parameters*
- *HIVE does not protect those pages that are accessible without parameters. This allows HIVE to function without the configuration of Start Pages.*
- *HIVE does not protect cookies.*

All Configurations are split broadly into two areas - the general parameters which apply to the entire application, and the lower level parameters that apply to individual pages and fields. Each Configuration is tied to one or more domains (or IP addresses) in order to indicate to HIVE which HTTP traffic is to be protected.

Note that all HTTP traffic for unprotected domains is allowed through HIVE untouched and uninspected, as is all non-HTTP traffic by default. It is, however, possible to have HIVE block all non-HTTP traffic, a sensible setting when there is only a Web server behind the HIVE appliance.



Figure 4 - HIVE: Global settings

At the global level, the administrator can enable or disable various “sanity checks”, including:

- **Shell code detection** - This screens the content of the requests for binary characters.
- **SQL Injection detection** - This causes HIVE to review all incoming variables, searching for unexpected SQL commands.
- **Cross Site Scripting detection** - This allows HIVE to detect malicious HTML tags in URLs, form fields, etc.
- **Cookie Protection** - This has three levels: *No*, *Medium* and *High*. The first level does not check cookie values at all. The second and third levels both sign the content of cookies and validate the client requests.

With the *Medium* setting, however, only the cookie is eliminated (and the error logged) if its content is invalid - the actual request is allowed through. The *High* setting causes the request to be blocked completely, and the error logged.

- **Limit size of input** - This allows the administrator to predetermine the maximum size a text input, a text field (text area) and the maximum size of an uploaded file, if not already specified within the application. This option also allows HIVE to log whenever these default values are used (which can provide hints on where an application's own field-level validation needs to be tightened up).

Other options, which are intended to counter reconnaissance attempts, are *Display Date*, which allows the administrator to choose whether the Web server sends the *Date* header or not, and *Display Banner*, which allows the administrator to substitute the default server banner with a customised one to hide the identity of the Web server.

HTTP methods can also be restricted (GET and POST are the only ones set by default), and HTTP header sizes can be restricted to specified limits in an attempt to frustrate buffer overflow attempts. One other feature we would like to see here is the ability to strip out all source code comments, the source of a great deal of free information for the resourceful hacker.

Normally, the user who violates HIVE rules is presented with a HIVE error page instead of their expected results. However, it may not always be politic to make the hacker aware of the protection method being used, and so HIVE also provides the means to replace the default error messages with custom text, or to redirect the user to a different Web page (or site!) transparently.

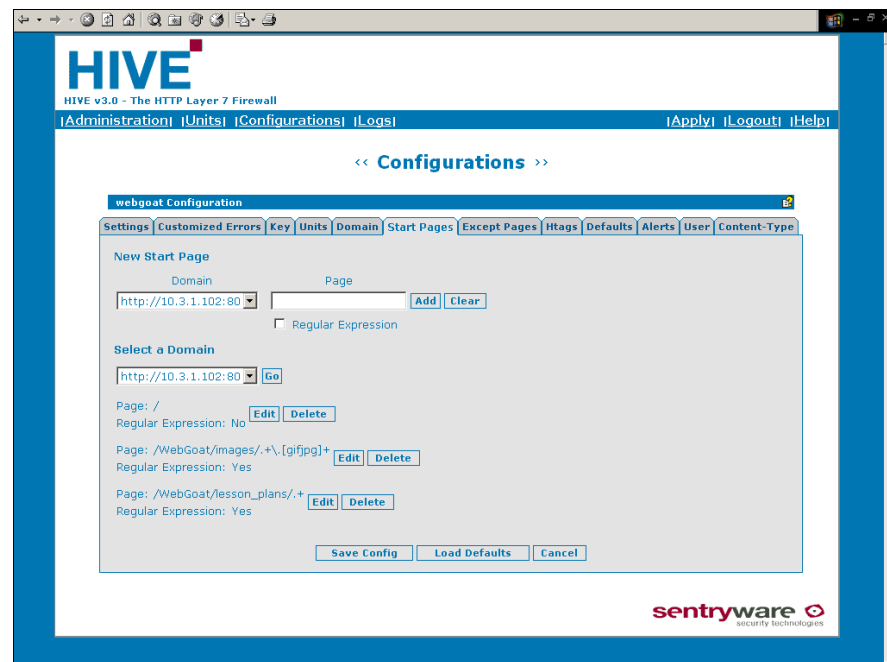


Figure 5 - HIVE: Defining Start Pages

Once the global parameters have been set, the administrator needs to define the lower-level elements of the application. Each application is effectively defined by its legitimate entry points - known as *Start Pages*.

Start Pages do not require a signed request to access them, and there must always be at least one Start Page, from which it is possible to access the rest of the Web application.

The pages which a user typically accesses directly (by entering the URL into the address bar of their browser) need to be configured as Start Pages - this may often be restricted to the home page of the Web site, or the main menu page of the application itself. It is possible to define Start Pages explicitly (using a complete URL) or via regular expressions (which allows a group of pages to be specified with a single setting).

Once a user has accessed a Start Page, HIVE will kick in with its Context Authentication process and everything from that point on is fully protected. The user will only be able to follow authenticated links from the Start Page to subsequent pages within the application, and any attempt to access a non-Start Page directly will be rebuffed.

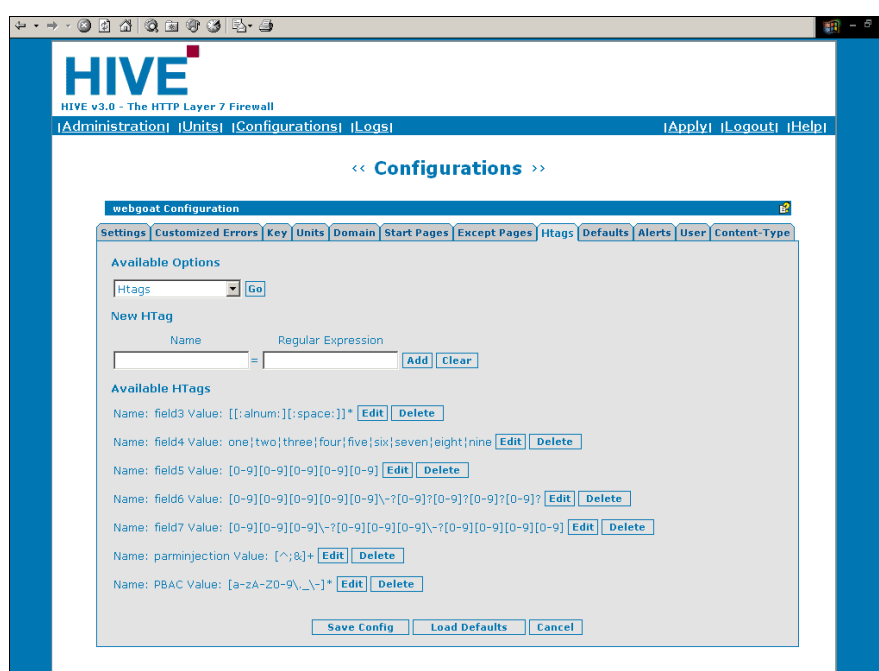


Figure 6 - HIVE: Defining HTAGs

Defining additional Start Pages at deeper levels of the application is thus to be discouraged, since that could provide “back doors” allowing a user to bypass application logic on pages above them. It is also important to note that Start Pages are the only pages where access is not restricted by HIVE, and therefore it is not recommended to install an application as a Start Page as it would not be protected against attack.

In some cases, a single Start Page may be all that is required to define a Configuration. Although this concept seems simple, it is made possible only by the Context Authentication capabilities of HIVE - once one entry point has been defined, the transparent protection is applied to every object which can be accessed from that point onwards.

It is worth taking a moment to consider the ramifications of this particular protection model. Not only is it impossible to access an unauthorised Web page directly, but it is impossible to access **any** Web resource directly when it is a part of the protected application.

For example, a typical directory traversal attempt followed by running `CMD.EXE` to subvert the OS would **also** be prevented, since `\windows\system32\cmd.exe` is not defined as a Start Page. This is how - despite the lack of typical IPS-style HTTP exploit signatures - HIVE was able to achieve a PASS in all our *Common Exploits* tests (*Test 1.1.31*).

Of course, not every Web application can be defined via a single Start Page, and the problems that can ensue should the administrator make a mistake in defining an application could have serious financial repercussions in some cases, should prospective customers be denied access to an important application.

Thus it is possible to put HIVE into *Learning Mode*, where HIVE performs all its normal Content Authentication functions but does not actually block any malformed requests. Instead, it simply creates log entries for each of them, allowing the administrator to use the information contained within logs to configure HIVE in a more secure and efficient way.

If Start Pages define the broad circle of application protection, HTAGs provide the administrator with the means to get down to the real detail.

Most applications will, at some point, ask the user to enter information with complex values (such as e-mail address, telephone numbers, social security ID numbers, and so on) which HIVE cannot predict until the client sends the request to the server.

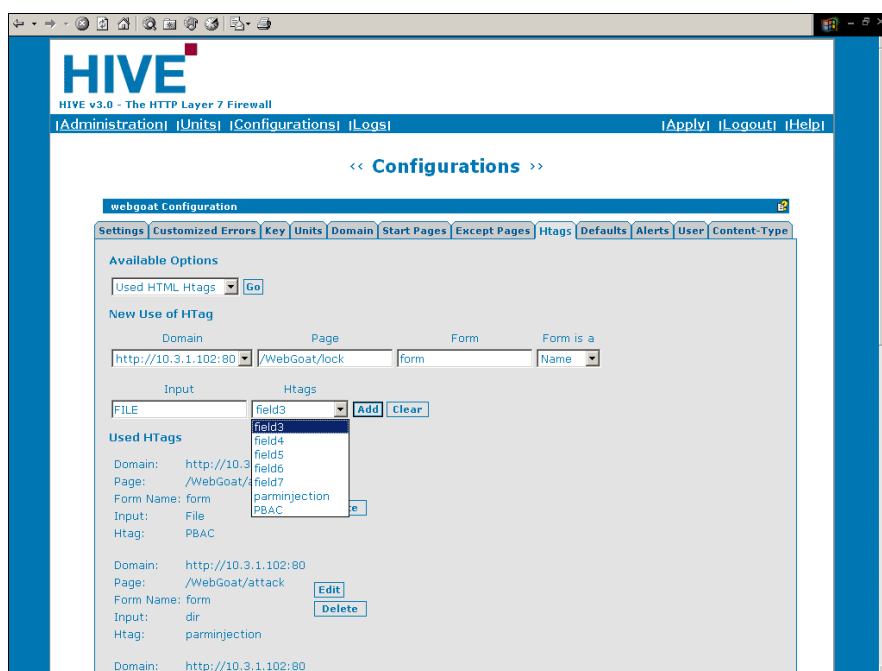


Figure 7 - HIVE: Using HTAGs

HTAGs permit the administrator to define the format of these values using regular expressions. Those HTAGs can then be applied to individual fields on a form (or to a cookie), for example, allowing HIVE to enforce those values before the request is passed. This means that applications that rely on client-side field validation (which can be subverted using Web proxy tools) can have that validation reinforced and re-applied once the request has left the client (and once it is beyond the influence of the hacker's tools).

One shortcoming we noted here was an inability to negate an entire regular expression (i.e. to specify that a particular value is **not** found in a data field). A “negate” checkbox for each HTAG is to be included in a future release.

HIVE also allows applications to interact directly with HTAGs. For this the designer of the page only needs to know the HTAGs that have been defined within HIVE, and he can then include in the INPUT field of the form `HIVETAG="PHONE-NUMBER"`. HIVE then captures this and knows that the input to this form should be protected by the HTAG that enforces a telephone number format. However, in this case HIVE can carry out this check *prior* to the request reaching the application, freeing the designer from having to worry about security within the application itself.

Alert Handling

HIVE has a monitoring system, which displays the errors that are produced during the communication between the Web client and the HTTP server.

Providing the application has been defined correctly by the administrator in the first place, it is important to note that illegal access attempts blocked by HIVE are *real* attempts - genuine modifications of requests requiring an effort on the part of a malicious user. Unlike generic network level signature patterns which can be triggered by innocent network traffic, once an application has been defined accurately, HIVE is incapable of producing false positives.

CONFIGURATION	DATE	IP	SERVER	DESCRIPTION	URL
NSS_Homepage	[12/Sep/2004:17:39:56 +0000]	10.3.1.104	http://10.3.1.91	Signature not found while trying to access a non-start page Add as startpage?	/cgi-bin/php?Qallas=%Dacate&sc=password
NSS_Homepage	[12/Sep/2004:17:39:58 +0000]	10.3.1.104	http://10.3.1.91	Signature not found while trying to access a non-start page Add as startpage?	/cgi-bin/php?Qallas=%Dacate&sc=password
NSS_Homepage	[12/Sep/2004:17:52:23 +0000]	10.10.1.1	http://10.3.1.91	Signature not found while trying to access a non-start page Add as startpage? <i>(Wow, this is a startpage)</i>	/download/download.htm
NSS_Homepage	[12/Sep/2004:17:52:33 +0000]	10.10.1.1	http://10.3.1.91	Signature not found while trying to access a non-start page Add as startpage?	/articles/articles.htm
NSS_Homepage	[12/Sep/2004:17:54:33 +0000]	10.10.1.111	http://10.3.1.91	The signature was modified by the client	/labs.htm
NSS_Homepage	[12/Sep/2004:17:58:10 +0000]	10.10.1.111	http://10.3.1.91	The signature was modified by the client	/security.htm
NSS_Homepage	[12/Sep/2004:17:58:33 +0000]	10.10.1.111	http://10.3.1.91	Signature not found while trying to access a non-start page Add as startpage?	/contct.htm
NSS_Homepage	[12/Sep/2004:17:58:54 +0000]	10.10.1.111	http://10.3.1.91	Signature not found while trying to access a non-start page Add as startpage?	/ProductTesting/ProductTesting.htm

Figure 8 - HIVE: Viewing log entries

There are three different ways to access the HIVE log information:

- *Directly, using the Web configuration interface*
- *Via e-mail*
- *Using SYSLOG*

Within the Web interface, the log entries are displayed on a page where the administrator can elect to view the last 25, 50, 100 or All alerts. The log entries are displayed in a simple HTML table with the following columns:

- **Configuration** - The application Configuration that triggered the alert. The administrator has access to all log entries from all Configurations. The user of a particular Configuration will see only the log entries for that application
- **Date**
- **IP address** - The source of the alert
- **Server** - The target application server
- **Description** - Detailed description of the event which triggered the alert
- **URL** - The URL which triggered the alert.

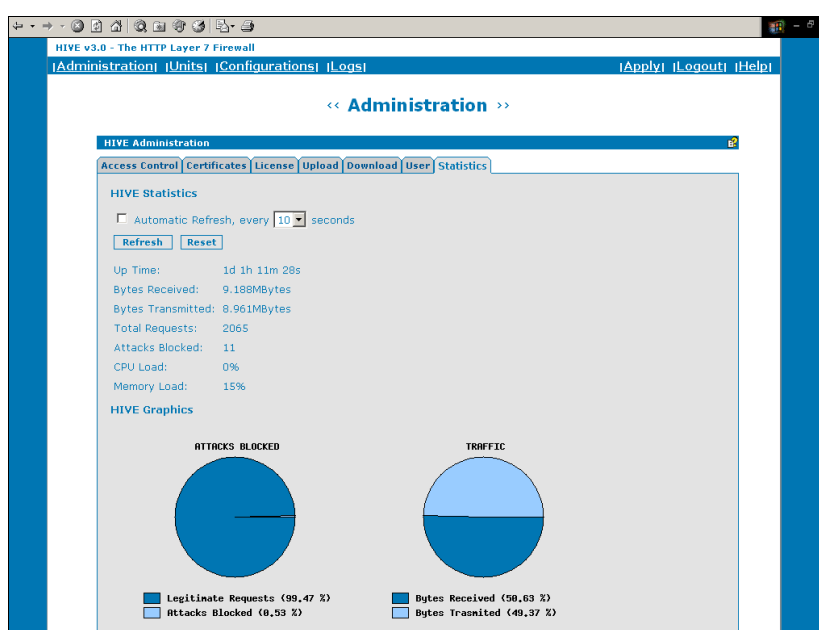


Figure 9 - HIVE: Statistics summary screen

Because the alerts are displayed in a simple table, there is no way to sort the entries by a different column, nor to filter them to display, for example, all the alerts for a particular Configuration.

Neither is it possible to acknowledge individual alerts and clear them from the list - it is only possible to clear the entire list or nothing at all. It is thus not as easy as it should be for the administrator to process the log entries. The only nod towards allowing further analysis to take place is the ability to download the current log as a text file, which could then be read in to another application perhaps.

The one convenience feature which is supplied via the logs is the ability to define the URL mentioned in the alert as a Start Page with a single mouse click. This works well when HIVE is in Learning Mode, where it will generate a large list of "suspicious" URLs, allowing the administrator to choose those which are legitimate Start Pages and add them to the Configuration relatively quickly. Each alert modified in this way turns green on-screen, but it is not possible to filter those out of the log display, once again making it more difficult than it should be to process a large block of log entries in Learning Mode.

This is one area of the product which would benefit from further development.

Reporting and Analysis

Within the HIVE Administration menu there is a Statistics option which provides a summary screen containing the following information:

- **Up Time** - Time since the last reboot.
- **Bytes Received** - The number of Bytes received.
- **Bytes Transmitted** - The number of Bytes transmitted.
- **Total Requests** - Total number of requests processed by HIVE.
- **Attacks Blocked** - The number of incorrect requests.

Below this are two pie charts which display the percentage of attacks blocked and the total traffic passing through each Ethernet interface. Apart from this display, there are no reporting or analysis features built in to HIVE.

Verdict

Performance

HTTP performance with security policies enabled were felt to be within acceptable limits for a device of this type. Note, however, that performance takes a further hit when HIVE is forced to sign objects on a page.

In our “real world” tests, where genuine Web pages were mixed with associated GIF/JPG images to increase the overall response size, HIVE performs much better.

Basic latency figures were acceptable for a device of this type with all packet sizes at traffic loads up to 250Mbps. With more typical (larger) packet sizes, latency remained acceptable up to 500Mbps, and it was around 800Mbps before the device began to drop packets and latency increased to unacceptable levels. HTTP response times were acceptable across all tests up to 75-80 per cent of the maximum load (as determined in our tests).

Exposing the sensor interface to an extended run of ISIC-generated traffic had no adverse effect, and the device continued to detect and block all other exploits throughout and following the ISIC attack.

Some people reading this report may be alarmed at the apparently low figures reported during these tests. However, it is worth reiterating that this is NOT a network-level device, and its only requirement is to be able to outperform the Web server it protects and the Internet pipe to which it is protected.

Whereas HIVE would act as a bottleneck in front of a heavily-used quad-Pentium server installed on a Gigabit LAN, it should be perfectly capable of handling a typical Web server on a typical high-speed Internet connection. Careful capacity planning is required on the part of the administrator, however, particularly in view of the overall limit on HTTP connections per second.

Security Effectiveness

Most of our attempted attacks were blocked successfully with very little configuration required. One of the most striking benefits of HIVE was that there was no need to perform any changes on any of our applications in order for it to function, and a high degree of protection was provided with a minimal amount of configuration of the product.

Buffer overflow protection, shell code detection, Cross Site Scripting protection and Cookie Poisoning prevention all worked well.

Interestingly although HIVE does not contain signatures or a detection engine for common HTTP vulnerabilities, it still managed to detect and block all of the exploits we attempted either via the ability to detect shell code, or via the ability to block direct access to disallowed pages (including CMD.EXE and other critical system files).

Overall, we were very impressed with the high level of protection provided for such a minimum amount of configuration.

Usability

The HIVE Web interface is something of a mixed bag. Once the administrator is familiar with its operation then it is certainly very responsive and quick to use. Day to day operations can be performed very quickly, despite the overall complexity of the product (Web Application Firewalls in general are not tools which are suitable for the beginner).

However, we found it to be very counter-intuitive in several places, particularly when first creating a Configuration - a process which in our opinion is crying out for a hand-holding "Wizard" capability to make life easier for the administrator.

There is no central management server, and no easy way to create multiple policies for separate devices and deploy them from a single console. Alert handling capabilities are extremely basic, and there is virtually nothing in the way of management reporting and analysis.

It could be argued that the Web interface currently does everything that it **needs** to in order to make HIVE an effective security tool, but there is definitely considerable room for improvement.

None of this affects the overall ability of HIVE to perform its allotted tasks, however - as a Web Application Firewall, HIVE does an excellent job.

Contact Details

Company name: SENTRYWARE

E-mail: info@sentryware.com

Internet: www.sentryware.com

Tel (Americas): (+1) 877 780 4864

Tel (EMEA & International): (+34) 902 412 422

APPENDIX A – TEST RESULTS

The aim of this procedure is to provide a thorough test of all the main components of a Web Application Firewall 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 a mix of Allied Telesyn AT-9816GB and AT-9812T 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 an in-line device during testing (if possible). 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 external 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 (where available) 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 attacks accurately, whilst remaining resistant to false positives. All tests in this section are completed with **no background network load**.

Test 1.1 - Attack Recognition

Whilst it is not possible to validate completely the entire range of attacks which can be detected by any sensor, this test attempts to demonstrate how accurately the sensor detects and blocks a wide range of common attacks as detailed in the *Open Web Application Security Project (OWASP) Top Ten Most Critical Web Application Security Vulnerabilities* (www.owasp.org). All attacks are run with no load on the network and are not subjected to any evasion techniques.

To demonstrate these vulnerabilities and provide a vulnerable application for the device to protect, we use a mixture of applications developed in-house specifically for this test, and the WebGoat application. WebGoat is a full J2EE web application designed to teach web application security lessons, and is available from the OWASP Web site. Most of the tests we devised for this methodology can be simulated or approximated using WebGoat.

Our attack suite contains over 30 basic attacks (plus variants) covering the following areas:

Test ID	Attack Category	Test Description
1.1.1	Buffer Overflows	Attempt to overflow input field on form
1.1.2	Hidden Field Tampering	Change price in hidden field after SUBMIT
1.1.3	Cross Site Scripting (GET)	Attempt to display document cookie via script in URL
1.1.4	Cross Site Scripting (POST)	Enter script in input field on e-mail submission form
1.1.5	Parameter tampering	Alter target e-mail address on e-mail submission form after SUBMIT
1.1.6	Buffer Overflows	Restrict size of message returned in e-mail submission form
1.1.7	Input Validation	Change validated values after SUBMIT (product should enforce same field validation)
1.1.8	Injection Flaws	Enter ..\..\.<filename> when selecting from list of files on-screen
1.1.9	Broken Authentication	Remove parameter field after SUBMIT
1.1.10	Broken Authentication	Change USER or ROLE after submit
1.1.11	Cookie Poisoning	Add AuthCookie=***** to Cookie: Header
1.1.12	Cookie Poisoning	Modify Cookie after SUBMIT
1.1.13	Cookie Poisoning	Modify Cookie file after it has been stored on local hard drive
1.1.14	Cross Site Scripting (POST)	Enter script in message field on guestbook form
1.1.15	Cross Site Scripting (POST)	Enter script in USER field on guestbook form
1.1.16	Injection Flaws	Attempt local directory listing (concatenate OS commands when accessing local OS)
1.1.17	SQL Injection	Attempt to list all entries of database - enter SQL commands in URL
1.1.18	SQL Injection	Attempt to list all entries of database - enter SQL commands in input field
1.1.19	Broken Authentication	Remove PASSWORD field after SUBMIT (fail-open authentication)
1.1.20	Invalid request	Attempt a non-HTTP connection or chunked request
1.1.21	Invalid Request	Attempt a disallowed method (HEAD instead of GET)
1.1.22	Invalid Request	Enter invalid server ID in HTTP/1.1 HOST header field
1.1.23	Invalid Request	Submit request containing shell code
1.1.24	Forceful Browsing	Attempt to access disallowed Web page directly
1.1.25	Forceful Browsing	Attempt to access "sample" Web site directly
1.1.26	Forceful Browsing	Attempt to subvert application flow - change STEP number directly in URL
1.1.27	Forceful Browsing	Attempt to subvert application flow - change STEP number after SUBMIT
1.1.28	Parameter Tampering	Change submitted parameter directly in URL
1.1.29	Information Disclosure	Filter/replace server banners
1.1.30	Information Disclosure	Strip comments from HTML/Java code
1.1.31	Common Exploits	Attempt common HTTP exploits (test-cgi, PHF, etc.)

Each of these attacks are proven to be effective against the back-end applications, and we expect the device under test to prevent all of them and raise appropriate alerts. Tuning of the application policy and/or signatures is allowed in order to ensure that the device is able to protect the application completely.

Section 2 – Evasion

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

Test 2.1 - URL Obfuscation

A number of common HTTP exploits are launched whilst applying various URL obfuscation techniques made popular by the Whisker Web server vulnerability scanner, including:

Test ID	Test Description
2.1.1	<i>URL encoding</i>
2.1.2	<i>../ directory insertion</i>
2.1.3	<i>Premature URL ending</i>
2.1.4	<i>Long URL</i>
2.1.5	<i>Fake parameter</i>
2.1.6	<i>TAB separation</i>
2.1.7	<i>Case sensitivity</i>
2.1.8	<i>Windows \ delimiter</i>
2.1.9	<i>Session splicing</i>

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 – Performance Under Load (No Security Policies Applied)

The aim of this section is to verify the maximum raw processing performance of the sensor. This is achieved by measuring packet processing performance and HTTP performance with no security policies applied.

Test 3.1 - 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 (not port 80) 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.

Test ID	Test Description
3.1.1	<p>256 byte packets - maximum 453,000 packets per second</p> <p><i>This test measures packet processing performance under the most extreme conditions for a Web Application Firewall - it is unlikely that any real-life HTTP server will ever be expected to handle network loads of over 450,000 256-byte packets per second. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic</i></p>
3.1.2	<p>550 byte packets - maximum 220,000 packets per second</p> <p><i>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, however, and there should be nothing 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. Repeated with 250Mbps, 500Mbps, 750Mbps and 1000Mbps of background traffic</i></p>
3.1.3	<p>1000 byte packets - maximum 122,000 packets per second</p> <p><i>This test demonstrates the ability of the device to process large packets. This provides the easiest environment for the device under test - 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</i></p>

Test 3.2 - Maximum Capacity HTTP Traffic

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, 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 to determine the maximum capacity (connections per second and Mbits per second) at varying packet sizes and using varying sizes of HTTP response.

Each transaction consists of a single HTTP GET request (thus *connections per second = transactions per second* in all these tests) 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, though there are no embedded links to other pages or elements which need to be defined within a security context. This test provides an excellent representation of a live network (albeit one biased towards “sterile” HTTP traffic) at various network loads.

Test ID	Test Description
3.2.1	1000 byte packets - 44KByte response (31 packets per transaction)
3.2.2	550 byte packets - 22KBytes response (37 packets per transaction)
3.2.3	440 byte packets - 11KBytes response (19 packets per transaction)
3.2.4	360 byte packets - 5KBytes response (9 packets per transaction)
3.2.5	285 byte packets - 2KBytes response (4 packets per transaction)

Section 4 – Performance Under Load (Security Policies Applied)

The aim of this section is to verify the ability of the sensor to handle varying loads of HTTP traffic whilst being expected to inspect the contents of that traffic according to the security policies applied.

Test 4.1 - Standard Spirent Avalanche Traffic

The default traffic generated by the Spirent Avalanche/Reflector devices is very realistic and provides for very repeatable tests. However, it is also very “sterile” inasmuch as it does not contain a variety of elements which must be inspected and acted upon by the device under test.

These tests, therefore, are expected to provide the best results under all test conditions.

Test ID	Test Description
4.1.1	1000 byte packets - 44KBytes response <i>Although the packet size is large for this test, so is the response size. This represents a worst-case scenario for the device under test due to the amount of data that is returned for each transaction, all of which needs to be inspected</i>
4.1.2	550 byte packets - 22KBytes response <i>With smaller packets sizes than the previous test, the response size is still large enough to cause problems for the device under test.</i>
4.1.3	440 byte packets - 11KBytes response <i>This test demonstrates a good average packet size and response size</i>
4.1.4	360 byte packets - 5KBytes response <i>This test demonstrates a good average packet size and response size</i>
4.1.5	285 byte packets - 2KBytes response <i>With very small packets and the smallest of the response sizes, this test is a best-case scenario for the device under test</i>

Test 4.2 - NSS Home Page Only - No Images

In order to introduce a realistic workload for the device under test, we switch out the Spirent Reflector and replace with a high-specification Web server running IIS and containing a copy of the NSS Group Web site. The dual-processor Dell PowerEdge 1750 server is capable of 1Gbps throughput so as not to act as a bottleneck for the device under test.

Test ID	Test Description
4.2.1	440 byte packets - 11KBytes response <i>As with the tests in Section 4.1, each transaction consists of a single page. In this case the page is the home page of the NSS Group Web site. This page is 11Kbytes in size, and contains a total of 24 embedded objects and links to other pages which may be of interest to any Web Application Firewall's inspection process. This test represents a worst-case “real world” scenario for the device under test.</i>

Test 4.3 - NSS Home Page + Ten Associated Images

The aim of this test is the same as for Test 4.2.1, but this time we add a number of images to the transaction.

This is intended to provide a more realistic - and easier - test scenario for the device under test since it does not have to inspect or process the image data, and thus although the transaction response size is greater, the processing overhead is lower than the previous test.

It is not often that the device under test will be faced with a high load of transactions which consist *purely* of HTML or application code which all needs to be inspected and processed - most Web pages contain a mixture of data types. Thus, this test is intended to replicate a typical "real world" situation.

Test ID	Test Description
---------	------------------

4.3.1	360 byte packets - 42KBytes response
-------	---

In this test each transaction consists of a single HTML page along with ten associated GIF/JPG images. As with Test 4.2.1, the page is the home page of the NSS Group Web site. This page is 11Kbytes in size, and contains a total of 24 embedded objects and links to other pages which may be of interest to any Web Application Firewall's inspection process. The transaction also contains a total of 31KB of images which do not need to be inspected or processed by the device under test.

Test 4.4 - Maximum Open Connections

It is important that the device under test cannot only support a sufficiently high rate of connection set-up and tear-down, but that it can also support a sufficiently large number of simultaneous connections. This test determines its maximum capacity.

Test ID	Test Description
---------	------------------

4.4.1	Maximum Open Connections
-------	---------------------------------

In this test the Spirent Avalanche is set to continually open HTTP connections with the Reflector. The Reflector is set to implement a delay on each transaction, thus ensuring that transactions remain open for a significant period of time (typical of a real world situation). The Avalanche is monitored and the point where transactions begin to fail is noted. Once the exact point of failure has been determined, the test is run for several hours with that number of transactions held open throughout.

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 an unacceptably high degree of latency on the packets passing through it, a network or security administrator would need to think carefully before installing such a device in front of a high capacity Web server or server farm.

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 ID	Test Description
---------	------------------

5.1.1	Device 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)

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 protection device, where failure can produce network outages.

Test ID	Test Description
---------	------------------

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 - this is intended to provide 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 device 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 attacks, raising an alert for each. Results are presented as a percentage of attacks blocked. If any recognisable attacks are passed - caused by either the volume of traffic or the sensor failing open for any reason - this will result in a FAIL

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 percentage of legitimate sessions allowed to pass. 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.

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 to the external network, and the ISIC target is on the internal network. ISIC traffic is transmitted through the sensor 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 firewall 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 ID	Test Description
----------------	-------------------------

7.1.1	<i>Open ports</i>
--------------	--------------------------

We will scan the open ports and active services on the management interface and report on known vulnerabilities.

7.1.2	<i>ISIC/ESIC/TCPSIC/UDPSIC/ICMPSIC</i>
--------------	---

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 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.

Sentryware HIVE V3.0 Test Results

Section 1 - Detection Engine

Test 1.1 - Attack Recognition				
Test ID	Attack Category	Description	Custom?	Pass/Fail
1.1.1	Buffer Overflows	Attempt to overflow input field on form	NO	PASS
1.1.2	Hidden Field Tampering	Change price in hidden field after SUBMIT	NO	PASS
1.1.3	Cross Site Scripting (GET)	Attempt to display document cookie via script in URL	NO	PASS
1.1.4	Cross Site Scripting (POST)	Enter script in input field on e-mail submission form	NO	PASS
1.1.5	Parameter tampering	Alter target e-mail address on e-mail submission form after SUBMIT	NO	PASS
1.1.6	Buffer Overflows	Restrict size of message returned in e-mail submission form	NO	PASS
1.1.7	Input Validation	Change validated values after SUBMIT (product should enforce same field validation)	YES	PASS
1.1.8	Injection Flaws	Enter ..\..\.<filename> when selecting from list of files on-screen	YES	PASS
1.1.9	Broken Authentication	Remove parameter field after SUBMIT	NO	PASS
1.1.10	Broken Authentication	Change USER or ROLE after submit	NO	PASS
1.1.11	Cookie Poisoning	Add AuthCookie=***** to Cookie: Header	NO	PASS
1.1.12	Cookie Poisoning	Modify Cookie after SUBMIT	NO	PASS
1.1.13	Cookie Poisoning	Modify Cookie file after it has been stored on local hard drive	NO	PASS
1.1.14	Cross Site Scripting (POST)	Enter script in message field on guestbook form	NO	PASS
1.1.15	Cross Site Scripting (POST)	Enter script in USER field on guestbook form	NO	PASS
1.1.16	Injection Flaws	Attempt local directory listing (concatenate OS commands when accessing local OS)	YES	PASS
1.1.17	SQL Injection	Attempt to list all entries of database - enter SQL commands in URL	NO	PASS
1.1.18	SQL Injection	Attempt to list all entries of database - enter SQL commands in input field	NO	PASS
1.1.19	Broken Authentication	Remove PASSWORD field after SUBMIT (fail-open authentication)	NO	PASS
1.1.20	Invalid request	Attempt a non-HTTP connection or chunked request	NO	PASS
1.1.21	Invalid Request	Attempt a disallowed method (HEAD instead of GET)	NO	PASS
1.1.22	Invalid Request	Enter invalid server ID in HTTP/1.1 HOST header field	NO	PASS
1.1.23	Invalid Request	Submit request containing shell code	NO	PASS
1.1.24	Forceful Browsing	Attempt to access disallowed Web page directly	NO	PASS
1.1.25	Forceful Browsing	Attempt to access "sample" Web site directly	NO	PASS
1.1.26	Forceful Browsing	Attempt to subvert application flow - change STEP number directly in URL	NO	PASS
1.1.27	Forceful Browsing	Attempt to subvert application flow - change STEP number after SUBMIT	NO	PASS
1.1.28	Parameter Tampering	Change submitted parameter directly in URL	NO	PASS
1.1.29	Information Disclosure	Filter/replace server banners	NO	PASS
1.1.30	Information Disclosure	Strip comments from HTML/Java code	N/A	FAIL
1.1.31	Common Exploits	Attempt common HTTP exploits (test-cgi, PHF, etc.)	NO	PASS ¹
Total				30/31

Section 2 - Evasion Techniques

Test 2.1 - URL Obfuscation				
Test ID	Evasion Technique	Detected?	Decoded?	Blocked?
2.1.1	URL encoding	YES	NO ²	YES
2.1.2	././ directory insertion	YES	YES	YES
2.1.3	Premature URL ending	YES	YES	YES
2.1.4	Long URL	YES	YES	YES
2.1.5	Fake parameter	YES	YES	YES
2.1.6	TAB separation	YES	YES	YES
2.1.7	Case sensitivity	YES	YES	YES
2.1.8	Windows \ delimiter	YES	YES	YES
2.1.9	Session splicing	YES	YES	YES
Total		9 / 9	8 / 9	9 / 9

Section 3 - Performance Under Load - No Security Policy

Test 3.1 - UDP traffic to random valid ports (not port 80)			
Test ID	Packet size	Packets per second (pps)	Max throughput
3.1.1	256 bytes	453,000pps	435Mbps
3.1.2	550 bytes	220,000pps	807Mbps
3.1.3	1000 bytes	122,000pps	883Mbps

Test 3.2 - Maximum capacity HTTP traffic							
Test ID	Packet Size (Bytes)	Response size (Mbytes)	Total no. response packets	HTTP response (ms)	Max conns per sec (cps)	Max packets per sec	Max throughput (Mbps)
3.2.1	1000	44	31	209	2000	94,000	775
3.2.2	550	22	37	205	2000	90,000	397
3.2.3	440	11	19	203	2000	52,000	197
3.2.4	360	5	9	202	2000	32,000	100
3.2.5	285	2	4	201	2000	20,000	49

Section 4 - Performance Under Load - Security Policy Applied

Test 4.1 - Standard Spirent Avalanche Traffic

Test ID	1000 byte packets - 44KByte response	25%	50%	75%	Max
4.1.1	Connections Per Second (cps)	21	42	63	84
	HTTP response time per transaction (ms)	210	210	210	980
	Throughput (Mbps)	8	16	24	38

Test ID	550 byte packets - 22KByte response	25%	50%	75%	Max
4.1.2	Connections Per Second (cps)	40	80	120	160
	HTTP response time per transaction (ms)	205	207	208	820
	Throughput (Mbps)	7.5	15	22.5	30

Test ID	440 byte packets - 11KByte response	25%	50%	75%	Max
4.1.3	Connections Per Second (cps)	65	130	195	260
	HTTP response time per transaction (ms)	204	205	205	520
	Throughput (Mbps)	6	12	18	24

Test ID	360 byte packets - 5KByte response	25%	50%	75%	Max
4.1.4	Connections Per Second (cps)	90	180	270	360
	HTTP response time per transaction (ms)	202	203	204	250
	Throughput (Mbps)	4	8	12	16

Test ID	285 byte packets - 2KByte response	25%	50%	75%	Max
4.1.5	Connections Per Second (cps)	125	250	375	500
	HTTP response time per transaction (ms)	202	202	203	220
	Throughput (Mbps)	3	6	9	12

Test 4.2 - NSS Home Page - No Images Loaded

Test ID	440 byte packets - 11KByte response	25%	50%	75%	Max
4.2.1	Simulated users	6	12	18	24
	Transactions per second (tps)	6	12	18	24
	HTTP response time per URL/page (ms)	800/800	950/950	1500/1500	1800/1800
	Throughput (Mbps)	1	1.5	2.5	4.0

Test 4.3 - NSS Home Page - Plus 10 Associated Images Loaded

Test ID	360 byte packets - 42KByte response	25%	50%	75%	Max
4.3.1	Simulated users	4	8	12	16
	Transactions per second (tps)	48	96	144	192
	HTTP response time per URL/page (ms)	400/2600	500/3400	700/4500	1000/6800
	Throughput (Mbps)	2	3.8	5.5	7.7

Test 4.4 - Maximum Open Connections

Test ID	Test Description	Result
4.4.1	Maximum simultaneous open TCP connections	4600

Section 5 - Latency & User Response Times

Test ID	Test Description	Packet Size	Network Load			
			250Mbps	500Mbps	750Mbps	1Gbps
5.1.1	Average latency (µs) with no background traffic	256	220.62	N/A ³	N/A ³	N/A ³
		550	268.71	259.98	287.84	N/A ³
		1000	303.19	318.60	345.61	N/A ³

Section 6 - Stability & Reliability

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

Section 7 - Management Interface

Test ID	Test Description	Result
7.1.1	Open Ports	PASS
7.1.2	ISIC/ESIC/TCPSIC/UDPSIC/ICMPSIC	PASS

Notes:

1. No common exploit vulnerabilities are included. However, once the application has been configured the user is not allowed to access any non-start pages directly - this prevented successful execution of all the common exploits we tried
2. Sees UUencoded traffic only
3. The device suffered large numbers of dropped packets at this load level, thus accurate latency figures were not possible

Section 1: Detection Engine

We installed one sensor in-line between our “internal” and “external” networks. This device was initially configured to protect all our “internal” Web servers in the following way:

- *The URLs used by the Spirent Reflector were designated as “start pages” using a regular expression to ensure that each page was inspected by HIVE (these pages contain no objects for HIVE to sign)*
- *The home page of our NSS test Web site used during the Avalanche performance tests was designated as a “start page”. This page includes a total of 24 objects which must be signed by HIVE*
- *The home pages of our vulnerable Web sites were designated as “start pages” - further configuration was required during testing of the vulnerable Web servers to ensure all tests could be completed*
- *Cookie protection was enabled*
- *Shell code detection was enabled*
- *Buffer overflow protection was enabled*
- *Banner replacement was enabled (all banners replaced with “HIVE Inspected”)*

Most of our attempted attacks were blocked successfully with no further configuration.

One of the most striking benefits of HIVE was that there was no need to perform any changes on any of our applications in order for it to function, and a high degree of protection was provided with a minimal amount of configuration of the product.

As expected, the most extensive configuration was for the enforcement of the field validation on our form (*Test 1.1.7*), where we needed to duplicate the client-side validation in regular expressions within HIVE. Once these had been applied, it was no longer possible to change the field values following submission of the form.

We were surprised that we needed to define regular expressions to catch directory traversals (*Test 1.1.8*), but once the Regex was in place it worked well. We feel this should be a global setting, however, like the shell code and cookie poisoning settings. A simple Regex was also required to prevent concatenation of OS commands (*Test 1.1.16*).

One thing which is not possible at the moment is to *negate* a regular expression in its entirety (i.e. to specify that a particular value or range of values should **not** appear in the data field). Sentryware is intending to include a *negate* checkbox in a future release.

Buffer overflows (*Test 1.1.6*) were caught via the global field length setting, but we found this to be too restrictive overall since it could be (as in our case) necessary to set this value too high to cope with one or two fields in the application. It would be preferable to allow the administrator to set this as a global “catch all” value, and then specify lower values where required on a field-by-field basis. Note that HIVE **is** capable of enforcing field lengths specified by the application programmer, however, which can have a similar effect. The *Shell code Detection* feature provides protection against HTTP requests containing binary characters.

We noted that Cross Site Scripting detection on POST requests concentrated on detecting the `<script>` tag in one of the fields posted - we would like to see that extended to check for other tags which can be used maliciously.

The only FAIL condition was an inability to strip out comments in source code, a feature which is simply not supported by HIVE at this point in time.

Interestingly although HIVE does not contain signatures or a detection engine for common HTTP vulnerabilities, it still managed to detect and block all of the exploits we attempted (*Test 1.1.31*) either via the ability to detect shell code, or via the ability to block direct access to disallowed pages (including CMD.EXE and other critical system files).

Overall, we were very impressed with the high level of protection provided for such a minimum amount of configuration.

Section 2: Evasion Techniques

Resistance to HTTP evasion/obfuscation techniques was excellent, with HIVE achieving a clean sweep across the board in our evasion tests. Every technique was detected and blocked effectively, and only one was not successfully decoded.

Section 3: Performance Under Load (No Security Policy)

UDP performance with a variety of packet sizes was poor with smaller packets (less than 512 bytes) and barely acceptable at larger packet sizes when compared to a pure networking device. However, since this type of device is intended to sit in front of a single Web server, or small group of Web servers, its overall packet processing performance should be considered acceptable. The bottleneck with this type of device is always likely to be the CPU rather than the network performance.

With pure HTTP traffic, and no security policies applied, the device could handle a maximum of 2000 new connections per second across all response sizes and all packet sizes, indicating that this is a memory restriction on the size of the connection tracking table rather than a processing limitation.

Whilst this is hardly blistering performance, it is adequate for the overall performance levels of the device once security policies are applied. Clearly, however, an administrator would need to consider this overall limit when deciding how many Web servers to place behind a single HIVE appliance.

Section 4: Performance Under Load (Security Policy Applied)

Web Application firewalls generally are far more limited in their performance than other network-level IPS systems, since they generally have a much heavier processing overhead. In addition, given their usual deployment position in front of a single Web server or small group of servers, their only real requirement is to be able to outperform the servers they protect in terms of connections per second and number of open connections supported.

The range of results demonstrated within *Section 4.1* were felt to be within acceptable limits for a device of this type. The device was configured such that each of the pages returned would have to be inspected, but there were no objects to be signed on any of these pages - thus, these represent the maximum performance figures that can be expected from HIVE at various packet and response sizes. As a general guide, a 5KB response is generally held to be the average on the Internet and thus *Test 4.1.4* is the closest approximation to the type of connection rates that could be sustained on a typical network.

Note that performance takes a further hit when HIVE is forced to sign objects on a page. *Test 4.2.1* substitutes the “sterile” Avalanche traffic with requests for the current NSS home page from our own Web site. This page is 11KB in size and contains 24 objects which need to be inspected and signed by HIVE. Maximum *transactions per second* and *overall throughput* both fall as a result.

However, this is a worst-case situation, since most pages have numerous associated objects and images which need to be downloaded at the same time as the HTML content, and these additional objects do not need to be inspected by HIVE.

Test 4.3.1 emulates this more normal “real world” state of affairs by having ten GIF/JPG images downloaded with our default home page.

With a total response size of 42KB, but still with the same 24 objects to sign, HIVE performs much better, achieving 192 transactions per second before response times become excessive and performance becomes erratic.

Note that we found the device could cope with short bursts of traffic well in excess of these figures - the only effect being a temporary increase in HTTP response times.

Some reading this report may be alarmed at the apparently low figures reported during these tests. However, it is worth reiterating that this is NOT a network-level device, and its only requirement is to be able to outperform the Web server it protects and the Internet pipe to which it is protected.

Whereas HIVE would act as a bottleneck in front of a heavily-used quad-Pentium server installed on a Gigabit LAN (perhaps supporting a large population of users, or many continuous concurrent transactions), it should be perfectly capable of handling a typical Web server on a typical high-speed Internet connection. As with all Web Application firewalls, careful capacity planning is required on the part of the administrator.

Section 5: Latency

Basic latency figures were just about within acceptable limits for a pure networking device with all packet sizes at traffic loads up to 250Mbps, ranging from 220µs with 250Mbps of 256 byte packets, to 303µs with 250Mbps of 1000 byte packets.

With more typical (larger) packet sizes, latency remained acceptable up to 500Mbps (260µs with 550 byte packets and 318µs with 1000 byte packets), and it was around 800Mbps before the device began to drop packets and latency increased to unacceptable levels.

With layer 7 devices such as HIVE, microsecond latency is less of an issue than with network-level in-line devices such as Intrusion Prevention Systems. Thus the latency figures seen here are well within acceptable limits for a device of this type.

Of more relevance are the HTTP response times, which were acceptable across all tests up to 75-80 per cent of the maximum load (as determined in our tests). Once the device is stressed beyond this level, however, response times degrade quite rapidly as CPU utilisation of the HIVE appliance approaches 100 per cent.

Thus, providing capacity planning is performed thoroughly to ensure the device is not normally pushed beyond 75-80 per cent of its maximum load determined in these tests, response times will remain acceptable across all packet sizes and HTTP response sizes.

Section 6: Stability & Reliability

The HIVE appliance performed consistently and completely reliably throughout our tests. Under extended attack it continued to block 100 per cent of attack traffic, whilst passing 100 per cent of legitimate traffic.

Exposing the sensor interfaces to an extended run of ISIC-generated traffic had no adverse effect, and the device continued to pass legitimate traffic whilst detecting and blocking all exploits throughout and following the ISIC attack.

Section 7: Management Interface

Open ports on the management interface are restricted to just one - TCP/443 (HTTPS) - and even this is not visible on port scans.

The extended ISIC attack against the management interface had no effect on the appliance and its ability to detect and block attacks or to pass normal traffic. The sensor continued to work perfectly throughout and following the ISIC attack, and there were no residual stability problems.