Global Information Assurance Certification Paper

How to use Hogwash to protect servers from internal attacks

GSEC Practical (v.1.4b)

By Mark Bunn

Table of Contents:

Abstract:

One of the largest challenges facing us today is protecting servers against attacks originating from within the local network. Many of the products available today are either very costly, are only available for a limited number of Operating Systems and/or require installing additional software onto servers or changing their configuration (which may introduce compatibility issues with other applications or cause resource problems).

This paper will cover one way to address these issues, by using Hogwash; which is freely available, can be customized to protect against vulnerabilities that are present on hosts regardless of OS, and requires no changes to the servers.

The goal of this paper is to provide detailed information on setting up and configuring a Hogwash system to protect servers from specific attacks.

Purpose, history of Hogwash:

Hogwash was written as a simple packet filter (called Scrub) in 1996 to protect a web server that was being taken over regularly that the patches to fix the vulnerabilities caused the software it needed to break. In 1999 the packet processing engine was replaced with SNORT to add the ability to write rules and use existing snort rules and was called SnortScrub and then renamed to Hogwash. Later as more features were being added to Hogwash, the SNORT engine was showing its weaknesses for doing heavyweight packet scrubbing and was replaced by a rewritten version of the original engine which is called the H2 engine.[i]

About Hogwash:

Hogwash is an IDS/packet filtering system that has the primary feature of filtering attacks from packets by inspecting their contents and not dropping all packets from the source address (which can be spoofed). This results in a system that has a lower risk of causing DOS attacks that can result from false positives.

Hogwash version 0.5 is currently in its infancy as this is the first version that uses the H2 engine instead of Snort. The H2 engine currently has only 17 options on which to build packet rules which can cause some frustration when writing rules (especially for udp packets).

Hogwash versions prior to version 0.5 use SNORT 1.8.6 as the engine which means that snort rules can be used directly but since this version may be a risk of recent SNORT vulnerabilities, including the following, its use is not recommended. A "buffer overflow has been found in the snort RPC normalization routines by ISS X-Force[ii]. This can cause snort to execute arbitrary code embedded within sniffed network packets."[iii] The originally tested version of Hogwash (0.4pre1) uses SNORT 1.8.6, which is vulnerable.

The current version of Hogwash has three modes that are determined by the placement on the network and some configuration differences. These are IDS Mode, Inline Scrubber Mode and Honey Pot Control Mode. IDS Mode and Inline Scrubber Mode have no configuration differences as the only thing that makes a Hogwash system an inline scrubber (aka packet filter) and not an IDS is the placement on the network.

The first mode is IDS Mode. In this mode the system is attached to a span or mirror port on a switch (or other network device that has this feature) so that the system will watch traffic as it passes this port. In this mode, since the Hogwash system has no control over routing in the network, the only advantage Hogwash has over a normal IDS is the ability to send resets to break the TCP session. This mode renders as useless the most important feature of Hogwash for this project which is the ability to block attacks. As an IDS system Hogwash does not currently compete with more mature products like SNORT, so this mode currently should be considered as a means to test Hogwash on a live network.

The second mode is Inline Scrubber Mode, which can be stealth (no IP stack) or normal. (Stealth is one of the key features of Hogwash, which is its ability to function without having a TCP/IP stack. In order to achieve this you recompile the kernel with support for your NIC cards but without TCP/IP networking support.) In this mode the system is normally placed between the boarder router and the firewall on an Internet connection. This causes the Hogwash system to act as a firewall / router on the network since all traffic travels through the system. An important thing to be aware of is that Hogwash works independently of IP Forwarding, what this means is if Hogwash is running and you have IP Forwarding enabled then both will forward the packet causing two packets out for every one in! In Inline Scrubber Mode Hogwash has the ability to stop attacks by sending TCP resets, dropping the packet, and/or logging the packet. In future versions, Hogwash will also be able to sanitize packets to remove only the portion that matches a rule without dropping the whole thing, thus causing

attacks to be converted into normal traffic; similar to the cleaning function of most Anti Virus products.

The third mode is Honey Pot Control Mode, in which you setup the Hogwash system to act as a router to send different types of attacks to different honey pot systems via the use of multiple NICs. This is the only mode that has configuration changes because it routes to honey pots that you have setup to have the same IP Address and MAC address as production servers. Therefore Hogwash will arbitrate the IP and MAC conflicts that would normally result. This mode is currently experimental.

In addition to the modes there are also three modules that add more functionality; ATS, WebUnique and DNSUnique.

ATS, which stands for All Traffic Summary, records basic header information about all packets into a separate log. This information can then be used to get a better understanding of traffic from a historical prospective. This feature is very similar to tcpdump except that ATS creates a separate log every hour and ATS is not as verbose. ATS can be enabled by un-commenting the module ATS section in the config file and adding the path to the logging directory. See the section on customizing the configuration for more information.

WebUnique and DNSUnique require MySQL support which is currently undocumented and was not tested as part of this project. The WebUnique module is a anomaly engine for web sites. WebUnique keeps a database (using MySQL) of web pages visited and logs any unique page hits. The idea here is normal visitors will usually hit the same pages over and over whereas an attacker would be the first client to request an unusual page. The DNSUnique module is essentially the same as WebUnique except that it looks at DNS queries.

In this paper we are using Hogwash in Inline Scrubber Mode and are deviating from the normal placement (outside the firewall) as shown in the *Placement of the Hogwash system* section, and because the system will reside on the internal network we will be giving both interfaces IP Addresses. The additional modules will not be used since they do not match the current goal of protecting against specific attacks.

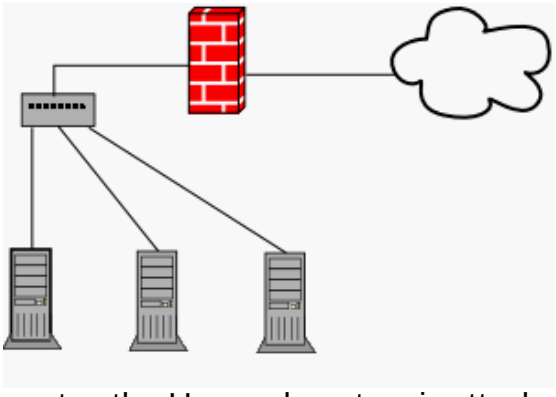
Placement of the Hogwash system:

Legend: The cloud in the following diagrams represents an INTERNAL network. The brick wall represents the Hogwash system.

In a single server environment (with the use of a crossover cable):

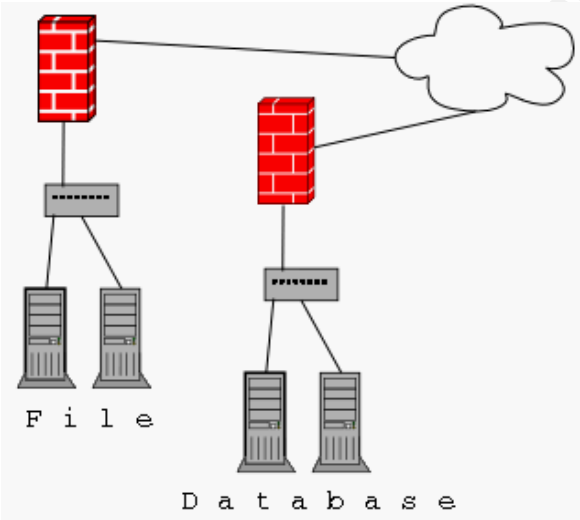
In this setup the Hogwash system is attached to the network where the server used to be and the server is connected to the second NIC in the Hogwash system via a crossover cable.

In a multi-server environment:



In this setup the Hogwash system is attached to the network where the servers used to be and the servers are either connected to a hub into the second Hogwash NIC or multiple NICs can be used in the Hogwash system.

In a grouped server environment:
(Where servers are grouped together by type of services provided and separate Hogwash boxes are used for each group):



In this configuration, rules are optimized by separating them onto multiple Hogwash systems to increase performance and by grouping servers according to the services offered to match the rules.

Operating System Distribution options: The following Operating System environments were used for testing since pre-compiled binaries of Hogwash are available or are able to compile Hogwash easily.

Trinux, which is available at http://trinux.sourceforge.net, is very lightweight and easily deployed. The system is booted from floppies or CD-ROM, which is then loaded into a ramdisk in memory. The results are lower latency times in execution at the cost of log space. Hogwash is available in the packages section

under “Get It”. This is currently the version of Hogwash that is vulnerable as described in the *About Hogwash* section.

RedHat Linux 9 at http://www.redhat.com, Debian 3.0 at http://www.debian.org. Both of these distros performed well. RedHat was quick and easy to install, but was more difficult to configure Hogwash when testing stealth mode. Debian required more attention during the install process, but was easer to setup and configure Hogwash.

Hardware and OS requirements:

Hardware: You will need hardware that will support 2 network cards and Linux. For this test an AST Pentium 90 with 2 3com 905 NIC cards and a Gateway Pentium 2 350 with 2 Intel Pro 100 cards were used. Initial setup is done with both NICs of the Hogwash system plugged into a hub or switch with one other PC and no connections to other networks.

RedHat: The only deviations from a default RedHat install are: On the firewall configuration screen choose Medium and customize to select both NICs as trusted. This is done so that iptables can be used to protect the machine if vulnerabilities are found in Hogwash in the future. If older hardware is being used you may also want to use LILO as the boot loader.
At the package selection screen choose either the minimal option (if available) or unselect all packages listed then check ‘**select individual packages**’.
Under **Development/Languages** select **gcc** and when prompted about package dependencies choose ‘ **Install packages to satisfy dependencies** ‘.

RedHat configuration: At this point, confirm both interfaces are working by typing **ifconfig** at a root prompt. Next, test connectivity by pinging both interfaces from another machine. Now its time to stop all the extra services from running at boot by issuing the command ‘ **chkconfig *name* off** ‘ where *name* is the service that we don’t want running.

To see what services were installed and are configured to start automatically type ‘ **chkconfig --list** ‘.

The following are the services I stopped: anacron, apmd, atd, autofs, gpm, irda, isdn, kdcrotate, kudzu, netfs, nfs, nfslock, nscd, pcmcia, portmap, rawdevices, rhnsd, saslauthd, sendmail, ypbind. For more information see Jay Beale’s article on killing daemons at http://bastille-linux.sourceforge.net/jay/killing-daemons.html.

This leaves sshd running for admin purposes. If you don’t use SSH then this can also be stopped as above. Otherwise add **ALL: ALL** to **/etc/hosts.deny** and add ‘ **sshd: *IP*** ‘ where *IP* is the IP Address of your admin machine to **/etc/hosts.allow**. Multiple IP Addresses can be listed with a space between them.

Installation of Hogwash:
Download the Hogwash tarball from http://hogwash.sourceforge.net/download.html and copy it to the Hogwash system using your favorite method… it even fits on a floppy! Once there unpack it using the command **tar –xvzf *packagename*** and cd into the newly unpacked directory until you see (ls) a file named configure. The tarball I downloaded had the files located in **./distro/devel-0.5/devel-0.5/**.
Now type **./configure** then **make**. You should now have a file named hogwash in this directory. Copy this file to a location in PATH: **cp hogwash /sbin** .
As a quick test type **hogwash** and you should see the following:

```
Hogwash H2 v2.1
by Jason Larsen

Useage:
----------------------------
hogwash <args>
  -c <Config File>
  -r <Rules File>
  -t  Parse Rules and Exit
  -n Process n packets and exit
  -l <Log Directory>
  -d Enter Daemon Mode (Background Execution)
```

Now we will need to create log and configuration directories, so
Type **mkdir /var/log/hogwash** and **mkdir /etc/hogwash**.
Copy the rules and config files to the /etc/hogwash directory.
cp *.rules /etc/hogwash and **cp *.config /etc/hogwash**

Just to make sure all is still well lets do another test:
Type **hogwash –c /etc/hogwash/stock.config –r /etc/hogwash/general.rules –l /var/log/hogwash**
The results should be:

```
Hogwash H2 v2.1
by Jason Larsen

Config file is /etc/hogwash/stock.config
Rules file is /etc/hogwash/general.rules
Log directory is /var/log/hogwash/
There is no MYSQL support
There is no MYSQL support
Loaded 14 rules
```

Hit **Ctrl + c** to quit.

Customizing the configuration:

Included in the Hogwash tarball is a stock.config file. Start by copying this file so that it is available for later reference.

cp /etc/hogwash/stock.config /etc/hogwash/live.config and edit the live.config file **vi /etc/hogwash/live.config**.

Here is a description of the sections and values:

-> The first section is labeled **system**; the Name and ID fields are for logging purposes only.

The **Threads** field controls how many CPU threads are launched for packet processing. One thread is given per interface regardless of this number. This option could be useful on multi-processor machines or on cluster systems that are thread based (MosiX for example). **Recommended value is 1**.

-> The second section is the **interface** section where each interface is described.

The **Type** field basically tells Hogwash what OS it is running on. There are values for bsd, OSX, Solaris and of course Linux.

The **Proto** field currently only has the option of Ethernet, but later releases of Hogwash are planned to support other protocols.

The **Role** field appears to only be used in conjunction with Honey Pot Control Mode to resolve MAC and IP Address conflicts and is not well documented as to what it really does so here is an excerpt from the online documentation "Role is used to give hints to Hogwash when there's things like IP address conflicts and MAC address conflicts."[iv]

Since we are using two interfaces for this project, we will uncomment the four lines of the interface eth1 section and remove the line **Role=Normal** above the </interface> line in the eth0 section. If *ifconfig* shows that your interfaces have names other than eth0 and eth1 then change the interface tags appropriately.

-> The third section is where **IPLists** are kept; these are used to define rules specific to servers by groups. This section was left as is.

-> The next section is where **actions** are defined. This tells Hogwash what to do when a rule is matched. The default action as defined will first display an alert on the system console, second place the alert into the file specified in parentheses, third copies the packet into the file specified in parentheses in tcpdump format and then drops the packet. We will add an action to test by logging and not dropping the packet by adding the following to our config file after </action>

```
<action test>
response=alert console
response=alert file(hogwash_test.alert)
</action>
```

-> The next few sections are for configuring additional modules like WebUnique, DNSUnique and ATS which were not used in this project. The *Unique module sections have the MySQL database parameters of dbase, which is the name of the database created for Web or DNSUnique with user and password being the MySQL credentials created to read and write records. The host parameter tells Hogwash the database server IP Address or hostname. The servers parameter tells Hogwash which IP Addresses Web or DNSUnique should monitor. The ATS module section contains one parameter: filename. The first part of this parameter should be changed to the path of the log directory like this ' filename=/var/log/hogwash/traffic_%y_%m_%d_%h.ats '. The time options of the filename are: %y for year, %m for month, %d for day and %h for hour. Minutes and seconds are not needed in the file name since a new file is created every hour. An interesting tidbit is that when Hogwash is executed with ATS enabled it immediately creates an ATS log file with the date 1969_12_31_18 until the rules are finished loading.

-> This brings us to the routing section which controls how packets are routed between the interfaces. There are five options to choose from; Sbridge, MacFilter, Broadcast, SIP and DIP. SBridge simply passes packets from one interface to the other and can be used with no more than two interfaces. In the online documentation it is mentioned that jabber conditions can occur when using SBridge, although this did not occur on the test network. MacFilter passes packets according to their MAC address so that broadcasts are not retransmitted out the same interface they came in on. Broadcast enables support for Ethernet broadcast packets. This is normally not needed since the OS kernel is able to handle the broadcast traffic. SIP which stands for Source IP, routes packets to interfaces according to their source IP Address. DIP which stands for Destination IP, looks at the destination IP Address and routes to an interface accordingly. Both SIP and DIP are most useful when using three or more interfaces for Honey Pot Control Mode. Be aware that non-IP packets are ignored and therefore not routed by SIP and DIP.

The two options that will work best for this project are SBridge and MacFilter. The other options are either to resolve specific problems or result in only TCP packets being passed so we will use the default option of MacFilter by un-commenting its line in the config file.

Now save and exit the live.config file.
See **Appendix A.** for an example live.config file.

Customizing rule sets (The fun starts here):

The steps used to include rules are:

1. Determine what services, their versions, patch levels and hot fixes are running on the server to be protected.
2. Determine what the vulnerabilities of these services are.
3. Determine what attacks could be used to exploit the vulnerabilities.

4. Find if other products (like Snort) have rules for these exploits and if so can we port these rules to Hogwash; if not do we know enough to write the rule ourselves.
5. Write the rule

1. There are various ways to determine what services are running, one of the quickest ways is to do a portscan of the server using nmap. Example: **nmap –sT *IPaddress*** then telnet to each port to see if banner messages are returned containing version information. Example: **telnet *IPAddress portnumber*** (You may need to hit enter a few times to generate a response). We'll use a Microsoft Personal Web Server as an example: nmap –sT iisserver returned port 80/tcp as open so now telnet iisserver 80, which returns Server: Microsoft-IIS/4.0.
2. Now we search for exploits using Google.com with the keywords " IIS 4 advisory ". This returns many results with one of the matches being "CERT Advisory CA-2001-13 Buffer Overflow In IIS Indexing Service "[v] which after reading the advisory it is determined that the iisserver box is vulnerable and cannot be patched (lets say it is imbedded in another product so that the released patch will not work).
3. Luckily this site also details how this exploit works. So we know that requests to .idq and .ida would be an exploit attempt since the website on this box does not use these file types.
4. The rules for this will be easy enough so we'll write it ourselves.
5. The rules

```
<rule>
ip dst(WebServers)
tcp dst(80)
tcp nocase(.idq)
message=%sip:%sp->%dip:%dp WEB-IIS .idq request
action=default
</rule>
<rule>
ip dst(WebServers)
tcp dst(80)
tcp nocase(.ida)
message=%sip:%sp->%dip:%dp WEB-IIS .ida request
action=default
</rule>
```

Alternately the ' I want to know if any attacks are attempted ' approach can be taken in which case all relevant rules will be used. This approach is useful when little is known about the server and time does not allow doing the research indicated in the steps above. This approach will result in much larger rules files which will create a slower system. Since in this project Hogwash is not being used as an IDS system but rather as a method to protect from specific attacks it is highly recommended to take the time to properly configure the rules.

Rule Porting:

The current version of Hogwash (0.5) no longer uses Snort so we cannot simply copy snort rules, although future versions are planned to have a Snort compatibility layer. Currently there are a limited number of tests that are built into Hogwash so we are limited as to what types of rules can be written. Hogwash is changing rapidly and the included tests will have additions soon, perhaps even by the time you are reading this paper.

As part of the Hogwash tarball you will have a file named stock.rules that contains the details on how to write Hogwash rules (see **Appendix B.**). One nice feature of the H2 engine is the simplicity of the rules in comparison to Snort rules.

We can use this information and cross reference it to write our rules from existing Snort rules. Information on writing Snort rules can be found at http://www.snort.org/docs/writing_rules/chap2.html. Snort rules can be found at http://www.snort.org/snort-db/.

As a process example I'll use the snort signature for Web .htr requests (SID 1619) which is:" alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"EXPERIMENTAL WEB-IIS .htr request"; flow:to_server,established; uricontent:".htr"; nocase; classtype:web-application-activity; reference:bugtraq,4474; reference:cve,CAN-2002-0071; sid:1619; rev:4;) "[vi]

First start with the example rule:

```
<rule>
ip  dst(WebServers)
tcp dst(80)
tcp nocase(cmd.exe)
message=%sip:%sp->%dip:%dp cmd.exe attempt
action=default
</rule>
```

Next change the message to contain the message from the snort rule:

```
message=%sip:%sp->%dip:%dp WEB-IIS .htr request
```

Now change the rule details to match the exploit shown in the Snort rule:

```
ip dst(WebServers)
tcp dst(80)
tcp nocase(.htr)
```

This gives us the completed rule:

```
<rule>
ip dst(WebServers)
tcp dst(80)
tcp nocase(.htr)
message=%sip:%sp->%dip:%dp WEB-IIS .htr request
action=default
</rule>
```

Testing to insure server availability:

Before implementing rules in a live environment change the action statement to log only by changing action=default to action=test, this will cause the rules to use the test action that we defined in the config file. After Hogwash has run for a period of time, check the log files to see if valid traffic has been falsely identified and change the rules accordingly. Test the rules again. Then change the action statement back to default.

Another step that can be taken is to attach the Hogwash system first to a span or mirror port on the switch of the server to be protected. If multiple machines are on this switch then set every IPList section of the config file to the IP Address of the server to be protected. As above let Hogwash run for an appropriate amount of time (a couple of days) and check the log files in /var/log/hogwash. When you are certain that Hogwash is performing as expected then proceed with your planned placement.

Setting up Hogwash as a Service:

Now that everything is working and has been tested we are ready to install Hogwash as a service on the machine.

To set Hogwash to start up whenever the machine is booted do the following:

1. Create a script that starts hogwash with the options you want: **vi Hog**, the completed script should look like this:

```
#!/bin/sh
#chkconfig: 2345 11 89
#description: Automates Hogwash packet filter
/sbin/hogwash -d -c /etc/hogwash/live.config -r /etc/hogwash/live.rules -l
    /var/log/hogwash
```

   The numbers following the chkconfig statement are; first the run levels (2345) to add the Hog script to and second, the order to start (11 th) and stop (89 th). By using these start and stop numbers the Hog script will start after networking and stop before networking.
2. Make the script executable: **chmod 700 Hog**
3. Copy the script to the /etc/rc.d/init.d/ directory: **cp Hog /etc/rc.d/init.d**
4. Add the script to start automatically using chkconfig: **chkconfig --add Hog**
5. Confirm that all went well by typing **chkconfig --list**, you should see Hog listed at the top with on next to 2 through 5.

References:

Dowd, Mark and Mehta, Neel. "X-Force Alerts and Advisories." 03 March 2003. URL: http://www.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21951 (5 March 2003).

Franz, Matthew. " Trinux Configuration." Trinux: A Linux Security Toolkit. Date Unknown. URL: http://trinux.sourceforge.net/install.html (Jan. 2003).

Unknown, Author. "Installing Red Hat Linux." The Official Red Hat Linux x86 Installation Guide. 2002. URL: http://www.redhat.com/docs/manuals/linux/RHL-8.0-Manual/install-guide/ (Jan. 2003).

Beale, Jay. "Killing Daemons!" Applied Operating System Security. 2000. URL: http://bastille-linux.sourceforge.net/jay/killing-daemons.html (Feb. 2003).

Karney, Charles. "SSH / Unix Security Tutorial." 12 Oct 1999. URL: http://w3.pppl.gov/~karney/ssh-sem.txt (Feb. 2003).

Larsen , Jason. "Setting up a Hogwash Box." Documentation. 6 July 2001. URL: http://hogwash.sourceforge.net/docs/setting.html (Mar. 2003).

Larsen , Jason. "Overview." Documentation. 6 July 2001. URL: http://hogwash.sourceforge.net/docs/overview.html (May. 2003).

Fyodor. " Nmap network security scanner man page." 2003. URL: http://www.insecure.org/nmap/data/nmap_manpage.html (Mar. 2003).

Havrilla, Jeffrey S. " CERT Advisory CA-2001-13 Buffer Overflow In IIS Indexing Service." CERT Advisories. 17 Jan 2002. URL: http://www.cert.org/advisories/CA-2001-13.html (Mar. 2003).

Unknown, Author. " Snort Signature Database SID 1619." Snort Signature Database. URL: http://www.snort.org/snort-db/sid.html?sid=1619 (Mar. 2003).

Unknown, Author. " buffer overflow." The Jargon Dictionary. URL: http://info.astrian.net/jargon/terms/b/buffer_overflow.html (Apr. 2003).

Glossary:

buffer overflow: What happens when you try to stuff more data into a buffer (holding area) than it can handle.

IDS: Intrusion Detection System

MosiX: A kernel patch and software application that make multiple PCs share the execution of processes without changing applications' code. It is available at www.mosix.org

Stealth (mode): Configuring a network device to not have an IP Stack; thereby making it more difficult to find on a TCP/IP network.

Tarball: A group of files that have been copied into a single file using the UNIX tar command.

Appendix A Example live.config file:

```
<system>
Name=Hogwash Sensor
ID=1001
Threads=1
</system>

<interface eth0>
Type=linux_raw
Proto=Ethernet
Role=Normal
</interface>

<interface eth1>
Type=linux_raw
Proto=Ethernet
Role=Normal
</interface>

<IPList WebServers>
0.0.0.0/0
</list>

<IPList DNSServers>
0.0.0.0/0
</list>

<IPList FTPServers>
0.0.0.0/0
</list>

<IPList AllServers>
WebServers
DNSServers
FTPServers
</list>

<action default>
response=alert console
response=alert file(hogwash.alert)
response=dump packet(packet.log)
response=drop
</action>

<action test>
response=alert console
response=alert file(hogwash_test.alert)
```

```
</action>
<routing>
Sbridge(eth0, eth1)
</routing>

<mangling>
</mangling>
```

Appendix B. Rule Syntax:

```
<rule>
keyword(options)
message= items -> items text goes here
action=action
</rule>

Action defines the action to perform when this rule matches.
Actions are defined in the config file.
```

Keywords:

```
interface name(eth0, eth1, eth5-eth6)
ethernet src(01:02:03:04:05:06)
ethernet dst(01:02:03:04:05:06)
ethernet type(IP, ARP, 0804)
ip src(10.10.10.2, WebServers, 192.168.0.0/16, 172.12.34.24-
  172.12.34.55)
ip dst(10.10.10.2, WebServers, 192.168.0.0/16, 172.12.34.24-
  172.12.34.55)
ip proto(TCP, UDP, ICMP, IGMP, PIM, OSPF, 13-15)
ip ttl(1-5)
icmp code(6)
icmp type(4)
tcp src(80, 21-25)
tcp dst(80, 21-25)
tcp content(bind|00 00 0A|)
tcp nocase(cmd.exe|00 00 0A|)
udp src(53)
udp dst(32000-32999, 53)
dns numquestions(2-10)
```

Message items:

```
%sip  - Source IP
%dip  - Destination IP
%sp   - Source Port
%dp   - Destination Port
%y    - Year
%m    - Month
%d    - Day
```

%h - Hour
%min - Minute
%s - Second
%usec - Microsecond

---

[i] Paraphrased from the Hogwash Documentation, Overview section at http://hogwash.sourceforge.net/docs/overview.html

[ii] "ISS X-Force has discovered a remotely exploitable buffer overflow condition in Snort. Snort is an open source intrusion detection system. A buffer overflow flaw exists in Snort RPC preprocessing code that is vulnerable to attack."

http://www.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21951

[iii] News item dated March 3, 2003 on www.snort.org

[iv] http://hogwash.sourceforge.net/docs/interface.html

[v] http://www.cert.org/advisories/CA-2001-13.html

[vi] http://www.snort.org/snort-db/sid.html?sid=1619