Hogwash, a Snort-Based Gateway-IDS

Scrubbing the Network

Firewalling and IDS (Intrusion Detection Systems) are actually two distinct areas: the former allowing only specific packets and connections, and the latter recognizing potential attacks and informing the admin. As an IDS merely recognizes known attack patterns, it is up to the admin to close down any security gaps. But there are scenarios where this is impossible, and this is where IDS systems that clamp down on ongoing attacks come into their own. BY RALF SPENNEBERG

Hogwash [1,7] is an IDS that implements this functionality. The "History" boxout shows the development of this Gateway-IDS, and the remainder of this article will be dedicated to installing, configuring, and operating Hogwash.

A Flexible Packet Scrubber

Hogwash supports three distinct modes: IDS, Scrubber and Bait-n-Switch.

In IDS mode, Hogwash monitors network traffic and alerts the admin in case of malevolent packets. Although it cannot intercept packets, it can cancel the connection by spoofing TCP resets. Hogwash can be configured to monitor traffic on multiple interfaces.

In Scrub mode, Hogwash acts as a packet filter. It spoofs TCP resets, drops packets or modifies them to effectively repel any attack. Hogwash can use up to 16 network adapters and can forward packets between these adapters, if required, working as a bridge rather than a router.

The software runs transparently in promiscuous mode at link level. The sysadmin can even disable the operating system's IP stack, and will need to disable IP forwarding, as Hogwash assumes responsibility for this.

In the experimental Bait-n-Switch mode, Hogwash will protect production systems without repelling attacks. Instead, it forwards suspicious connections to a honeypot to allow for closer analysis: an attack on the honeypot will not impact the network.

Installation

Hogwash is quite simple to install. After downloading and unpacking the sources, follow the typical configure-make pattern:

Listing 1: Configuration file

```
01 <system>
02 Name=Hogwash IDS Gateway
03 ID=1001
04 Threads=1
05 </system>
06
07 <interface eth0>
08 Type=linux_raw
09 Proto=Ethernet
10 Role=External
11 </interface>
12
13 <interface eth1>
14 Type=linux_raw
15 Proto=Ethernet
16 Role=Internal
17 </interface>
18
19 <IPList WebServers>
20 5.0.0.1/32
21 </list>
22
23 <action log>
24 response=alert console
25 response=alert file(/var/log/
   hogwash/hogwash.alert)
26 response=dump
    packet(/var/log/hogwash/
   packet.log)
27 </action>
28
29 <action drop>
30 response=alert console
31 response=alert
    file(hogwash.alert)
32 response=dump
    packet(packet.log)
33 response=drop
34 </action>
35
36 <routing>
37 MacFilter(eth0, eth1)
38 </routing>
```

```
./configure
make
```

Hogwash runs on any Linux distribution, although it makes sense to remove the operating system's IP stack (see Figure 1) to harden the host against attacks. Of course, this prevents the administrator from using TCP/IP to access the Hogwash host, and means resorting to administering the host locally.

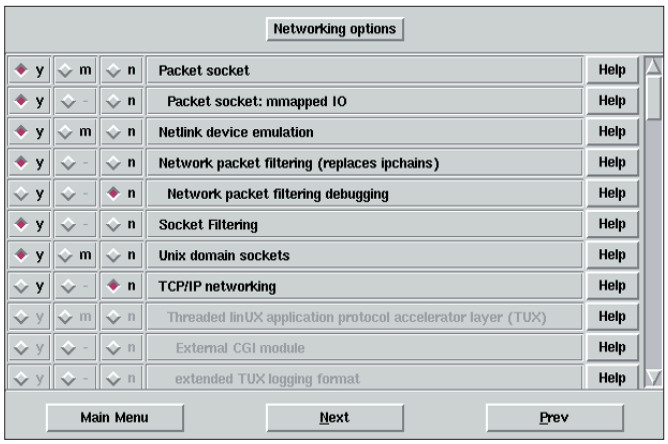

Figure 1: The kernel does not require an IP stack, if you use Hogwash. You can completely disable the stack allowing Hogwash to provide packet analysis capabilities

Scrubber Configuration

A configuration file and a ruleset are used to specify the role Hogwash plays. The package contains sample configuration files called *stock.config* and *test.config* and a number of sample rulesets:

```
general.rules
jason.rules
nikto.rules
nimda.rules
stock.rules
test.rules
x11.rules
```

The *stock.rules* ruleset also loads four files: *nikto.rules*, *x11.rules*, *nimda.rules*, and *general.rules*.

To configure the host, the administrator must first define the local system, the network interfaces, IP addresses, actions, and routing options. Listing 1 shows a sample file for the configuration shown in Figure 2. First, Hogwash is assigned a name and ID in the *<system>* area. The *Threads* statement specifies whether Hogwash will be launched in single-threaded mode (0), or assign a thread to each network card. Note that multi-threaded mode offers far superior performance. The network adapters are configured in the next step. Hogwash can use a variety of communication modes to talk to its network interfaces: Linux (*Type = linux_raw*), OpenBSD (*obsd_bpf*), MacOS X (*osx_bpf*), and Solaris (*solaris_dlpi*). The system can also read network packages from a tcpdump file in offline mode (*Type = tcpdump*). Hogwash supports only the Ethernet protocol, *Proto = Ethernet* and will not run on Token-Ring networks. The *Role = ...* parameter is optional and is used for routing and ARP.

To save time later when defining the ruleset, it makes sense to define IP address lists in the configuration file. The lists can contain individual IP addresses or complete networks in CIDR notation (Classless Internet Domain Routing). You can additionally define the actions Hogwash should perform, if an attack is detected:

- *response = alert console* – display the packet on the console.
- *response = alert file(filename)* – write the alert to a file.
- *response = dump packet(Filename)* – store the packet for later analysis.

If you want Hogwash to drop the packet, and thus protect a machine, you should stipulate *response = drop*. Rules will later access groups of actions to provide for appropriate responses.

If you are using Hogwash as a Gateway-IDS (also known as an Intrusion Prevention System), it will need perform packet forwarding. Hogwash refers to this as routing, although it is actually bridging. In addition to a simple bridge mode, *SBridge*, Hogwash also provides a flood protection variant, *MacFilter*. The system can only forward specific IP addresses via individual interfaces, depending on the source IP *SIP*, or destination IP *DIP*. The bridge function facilitates deployment, as Hogwash will run immediately on launching without any additional IP configuration. The Gateway-IDS is completely transparent at IP level.

Strictly by the Rules

The ruleset shown in Listing 2 comprises a selection of sample rules. Each entry can handle any given aspect of the packet: the network interface, Ethernet type (IP, ARP, and so on), IP addresses, IP protocol, ICMP code and type, UDP and TCP ports, and TCP packet content. You can specify messages and actions as a response to the packet: When defining a message (*message =*), the admin can use variables to access vital packet attributes and the current date. *action* refers to the actions defined in the configuration file (see Listing 1).

Listing 2 shows two simple sample rules. The first rule checks whether a TCP packet has been sent to port 80 and contains the *chunked* string. This is indicative of an attack on the Apache chunked encoding memory corruption vulnerability [4]. The second rule checks for ICMP echo requests. In this case, Hogwash will simply log the packet.

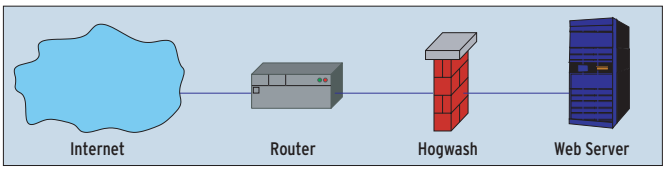

Figure 2: Hogwash typically assumes the role of a bridge: neither routers, nor webservers will notice the additional hurdle in this scenario. The filter will block a connection if an attack is detected

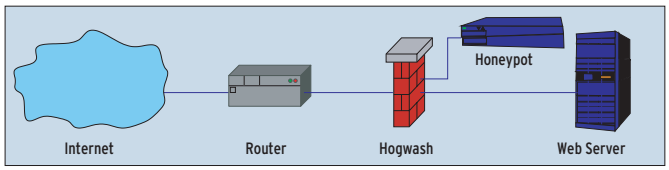

Figure 3: In Bait-n-Switch mode Hogwash protects a production server and re-routes attacks to the honeypot. This allows the admin to monitor the attacker at leisure without endangering production systems

The extremely comfortable rule syntax allows you to define arbitrary rules. A compatibility mode for Snort rule syntax is planned for future Hogwash versions. This would allow you to import Snort rules directly.

Bait-n-Switch

Bait-n-Switch mode allows you to deploy a honeypot to protect a production server. The Hogwash host forwards any traffic between the network interface, the honeypot, and the production server, as shown in Figure 3. Listing 3 shows an excerpt from the required script.

The admin first uses IP lists to specify those systems that Hogwash should always allow to pass through to the server <*IPList green*>, and those that will be sent off to the honeypot <*IPList red*>. After this step, the admin selects *Route* as the response to the rules, to forward the connection to the honeypot. The action provides a timeout and the IP addresses that Hogwash should never forward to the honeypot (*response = bns(Timeout,address list)*). The next step is to define which network interface is responsible for what:

```
bns(Internet,Production,
Honeypot,red_list)
```

The honyepot configuration should be identical to that of the production server. Most importantly, the IP addresses must match. The data provided by the honeypot should be identical to that on the genuine server to confuse the attacker.

Conclusion

Hogwash is an easy-to-install, invisible Gateway-IDS that operates on Layer 2 and is thus invisible to IP traffic – attackers will be unable to use *traceroute* to recognize the Hogwash host. The operating system in use does not even require an IP stack (see Figure 1). Hogwash is suitable for use as a central component in a Bait-n-Switch system that will distract attackers away from production systems to a honeypot, allowing the admin user to monitor the attacker's nefarious activities and the attack method offline at a later time.

Snort-Inline [5] is a similar Gateway-IDS – although it is harder to set up. It uses Lennert Buytenhek's bridging patch to forward packets via the Iptables *QUEUE* target to Snort-Inline. The software then tests the packets, only permitting harmless packets to pass. We will be looking at Snort-Inline in a subsequent issue. ■

Listing 2: Rule file

```
01 <rule>
02 ip dst(WebServers)
03 tcp dst(80)
04 tcp nocase(chunked)
05 message=%sip-%dip Apache
    Chunked Encoding Attack
    Bugtraq 5033
06 action=drop
07 </rule>
08
09 <rule>
10 icmp type(8)
11 message=%sip-%dip icmp echo
    request
12 action=log
13 </rule>
```

Listing 3: Bait-n-Switch Mode

```
01 <IPList green>
02 x.x.x.x/24
03 </list>
04
05 <IPList red>
06 y.y.y.y/32
07 </list
08
09 <Action Route>
10 response=alert console
11 response=alert file(
   /var/log/hogwash/hogwash.
   alert)
12 response=dump packet(
   /var/log/hogwash/packet.log)
13 response=bns(3600,green)
14 </Action>
15
16 <routing>
17 bns(eth0,eth1,eth2,red)
18 </routing>
```

INFO

[1] Hogwash: *http://hogwash.sourceforge.net*

[2] Snort: *http://www.snort.org*

[3] Ralf Hildebrand, "Cain and Abel – Snort and Nmap, two sides of the same coin": Linux Magazine Issue 4, January 2001, Page 46

[4] Apache vulnerability: *http://www.securityfocus.com/bid/5033/exploit/*

[5] Snort-Inline *http://www.snort.org/dl/contrib/patches/inline/*

[7] Securing an unpatchable Webserver – HogWash!: *http://www.securityfocus.com/infocus/1208*

THE AUTHOR

Ralf Spenneberg is a freelance Unix/Linux trainer and author. Last year saw the release of his first book: "Intrusion Detection Systems for Linux Servers". Ralf has also developed various training materials.

Snort's History

Way back in 1996 Jed Haile and Jason Larson had a problem: one of their webservers had a security hole, but it was tightly integrated with an additional software program that refused to run after they applied a security patch. Their solution was to write the predecessor to Hogwash, called Scrub.

The program filtered precisely those packets used for the attack on the webserver from the network traffic, allowing the admins to carry on running their webserver until they could port it to a new system.

Three years later the authors discovered Snort [2]. They were fascinated by its simple structure and easily intelligible rule syntax [3]. They added the internal Snort engine to Scrub and renamed their program, calling it Snortscrub. They finally renamed the product once more to Hogwash, intending to promote commercial use of the package.

A period of about two years, where development of the free version flagged, then followed. All the more so, since a whole range of new features was added to Snort in this period. But the Snort detection engine demonstrated some weaknesses.

So, Jason Larson (aka Anon Poet), decided to re-work the original Scrub detection engine and to revive it as H2. H2 has a compatibility mode that understands Snort rules.

Although work on compatibility mode is still in progress, the current version Devel-0.5 actually uses H2.