

Securing an Unpatchable Webserver... HogWash!

Jason Larsen 2001-07-31

Securing an Unpatchable Webserver... HogWash!

by *Jed Haile* and *Jason Larsen*

last updated July 31, 2001

The Problem at Hand

During a routine examination of a client's network we discovered a vulnerability on a Microsoft IIS 3 web server. After brief investigation, we discovered that this web server runs a mission-critical web application: it is the client's primary means of doing business and must be protected at all cost. The real problem is that this application is tightly bound to certain features of Microsoft's IIS 3 web server. We searched for a patch, but there were none. Microsoft's solution was to upgrade the server to a more recent version. We attempted to upgrade the server to IIS 4, but the result was disaster. A total rewrite of the web application using better technology is underway, but will not be complete for a long time. In the meantime the server needs to remain available and unhacked. What is the security professional to do?

The Limits of Intrusion Detection and Firewalls

The first step was to modify the Network Intrusion Detection system on the network to capture any attempts to exploit the vulnerability. This was easily done by adding a couple of rules to Snort. But this didn't solve the problem. We could catch an attacker, we could generate alerts, we could log packets, send e-mail, and call pagers. Nevertheless, the attacker could still get in, and by the time somebody could respond the damage would be done. We discussed the possibility of automatically updating router rules or firewall rules in the event of an attack, but this was not a good solution: a malicious attacker could spoof attacks from many sources and effectively deny everybody access to the server. This was considered to be an unacceptable risk.

A firewall would be of no help either. The web server had to remain available to the public and the vulnerability was in the web server software (IIS). A firewall has no way of determining if a request being sent to a web server is benign or malicious. So, while the firewall could stop traffic to ports that do not need to be publicly accessible, it was of no help in our situation.

This put us at zero for four: we couldn't patch, we couldn't upgrade, an IDS couldn't really help, and a firewall wouldn't add any protection. It was time to look for a new solution.

What we needed was a system that would do the job of both a firewall and an intrusion detection system, a system that could sit between the vulnerable system and the Internet watching for unacceptable packets and either sanitizing them or dropping them. In other words, an in-line packet scrubber. Like a firewall, this tool would have the ability to accept or deny packets. Like an intrusion detection system, it could examine a packet's contents, flags, headers, etc and make a decision. This is not a new idea, such devices have been called Gateway IDS, Intrusion Prevention Systems, Pattern Based Firewalls. There have been discussions on the Snort mailing list about building one, some vendors have claimed to be developing one. But none were immediately available. We would have to develop our own solution.

Building a Solution

The basic idea was to take Snort and enable it to drop or forward packets, depending upon a comparison of the packet and a set of pre-established Snort rules. Why start with the Snort IDS? It works well, is stable, scalable, and has a flexible rules language. Furthermore, it is well maintained, has a large community of users, and is easily extensible. Most important, the source code was freely available. This also helps make the setup, care, and feeding of the proposed tool much more accessible to all Snort users. We decided to call the tool HogWash, in honor of its Snort roots.

The HogWash system is placed in-line between the web server and the Internet. Functionally, HogWash is much like a bridging firewall, as it has two interfaces: inside and outside. The outside interface listens for packets destined towards the protected system. This is accomplished by putting the interface into promiscuous mode and capturing packets with libpcap. As packets are received they are passed to a modified Snort engine to make a forward/drop decision. If the packet is not dropped, a creative application of libnet enables HogWash to forward packets on towards the web server. Likewise, the inside interface is in promiscuous mode looking for outbound packets, which are then examined by the Snort engine. If the outbound packet is not dropped, it is forwarded by libnet through the outside interface. The use of promiscuous mode packet capture and libnet means that HogWash can run without an IP stack loaded on either interface, so it is nice and stealthy. It also means that no part of the packet is modified as it passes through HogWash. A properly configured HogWash box is almost completely invisible.

In the development of Hogwash, a new 'drop' ruletype was added to the stock Snort ruletypes of 'alert', 'log', and 'pass'. 'Log', 'alert', and 'pass' all behave like in they do in Snort. However, the 'drop' ruletype instructs HogWash to drop a packet that matches the designated signature. Plugins can be written to modify the packet and replace the original packet, or to do more advanced

inspection before making a drop decision.

Configuring HogWash to Close the Vulnerability

Next we had to build a ruleset to protect the web server from the exploit. The server wasn't running any other services, so we only had to actively protect the web service. The exploit that we were trying to protect against would be delivered as a GET request to the web server. The actual offending GET request could come in a large variety of forms, so we decided to configure HogWash to do a default deny on all incoming GET requests. Then we could explicitly tell HogWash what sort of GET requests to forward. To build an initial set of rules, we set up a standard Snort sensor to watch traffic to the web server and gave it one rule to begin with. The rule looked like:

```
alert tcp any any -> $WEBSERVER 80 (content: "GET"; nocase; msg:"Dropped GET request.");)
```

This rule caused an alert for every GET request to the web server. We sat and watched the alerts pile up for a few minutes and then stopped Snort. Each alert generated by Snort logged a packet that contained a GET request.

The next job was to characterize every valid GET request as a pass rule. For every valid GET request we added a pass rule to allow that particular request through. For example to allow people to get index.html we added:

```
pass tcp any any -> $WEBSERVER 80 (content: "GET "; nocase; content:"index.html"; nocase;)
```

We knew the exploit wouldn't work through the images directory so we exempted GET requests for the images directory:

```
pass tcp any any -> $WEBSERVER 80 (content:"GET"; nocase; content:"images/"; nocase;)
```

And onwards for every valid GET request. Ninety rules later we had a ruleset consisting mostly of pass rules, along with a few alert rules that nicely described the allowed functionality of the web server. We loaded the new rule set into Snort and watched traffic for a long while just to be sure that everything was kosher.

Next we placed the ruleset, with all the alert rules changed to drop rules, onto the Hogwash system and placed it in-line immediately between the switch and the web server. It was necessary

to put a crossover cable between the inside interface on the Hogwash system and the web server. We sat back and watched the alert file on the Hogwash system to be sure that things were functioning as we expected. Any packets that are dropped by HogWash are noted in the alert file. With this configuration, if a GET request that doesn't explicitly match one of the pass rules is sent to the web server, it will be dropped. So we had to be sure that everything was correct, or some part of the web site's functionality would be broken.

The advantage of this default deny approach is added peace of mind. It would be very difficult to come up with an exploit that would fit the allowed set of GET requests. The disadvantage is a tedious setup process and a relatively high maintenance cost. Anytime the web server's content is modified, new pass rules must be inserted into HogWash to make the content viewable. However, the investment in time is worthwhile if there is no other way to protect the server.

Other Benefits and Drawbacks

Hogwash is fully capable of using Snort's plugins. Of particular relevance to this application was the ability to use the Unicode decoder preprocessor and the HTTP decode preprocessor to decode all encoded GET requests before forwarding them onwards to the web server. This allowed us to write all of our rules using plain ASCII and not worry about someone getting clever with encoding and circumventing the filters.

When making decisions to drop or forward packets, users can't afford false positives. So extreme care must be taken in building the ruleset and lots of testing is highly recommended. It is useful to take advantage of many of Snort's advanced features to avoid false positives. Dynamic rules can be used to apply multiple checks to a packet before making a drop decision. Offset and depth keywords can be used to be sure that you are matching against the right content. Custom ruletypes can be defined to create an even more flexible scrubber.

It is also important to remember that a HogWash scrubber is an in-line device, so is a potential bottleneck. We have found that HogWash should scale to 100 megabit networks, but performance will vary according to your hardware and the ruleset that you use. A badly written rule could seriously disrupt a network. Be sure to test your configuration carefully before installing a HogWash scrubber on a production network.

Where to Get HogWash

HogWash is freely available under the GPL at <http://hogwash.sourceforge.net>. We encourage you

to download it and give it a try. We will gladly accept any bug reports, patches, or code. There is still a lot of work to do, HogWash is still in its infancy. As more users and developers jump on the bandwagon the HogWash scrubber should evolve into a powerful, flexible tool for protecting your network.

Relevant Links

[Hogwash](#)

[Snort](#)

[Snort.org](#)

[libpcap](#)

[tcpdump.org](#)

[libnet](#)

[The Packetfactory](#)

[Privacy Statement](#)

Copyright 2006, SecurityFocus