Global Information Assurance Certification Paper

An Analysis of the Snort Data Acquisition Modules

GIAC (GCIA) Gold Certification

Author: Chris Murphy, chrismrph0@gmail.com
Advisor: David Shinberg

Accepted:

Abstract

Snort is a commonly used open source Intrusion Detection System (IDS) with voluminous documentation and excellent community support. However, the data acquisition (DAQ) modules included with Snort IDS versions 2.9 and later are a relatively recent addition. DAQ allows new flexibility for Snort by separating the network capture functions out into external, loadable modules. DAQ also integrates inline intrusion prevention capability that was previously only available with add-on patches.

1. Introduction

Snort is an open-source Intrusion Detection System (IDS) that runs on Linux, UNIX, BSD variants and Windows. Martin Roesch created the Snort IDS software in 1998 and released it under the General Public License. Since then, it has become the most popular and widely used IDS software (Sourcefire, Inc).

Beginning with version 2.9, Snort uses a new mechanism for capturing packets. The Data Acquisition (DAQ) libraries were first announced in August 2010 on the VRT Blog (Combs, 2010). Previously, Snort integrated packet capture and other network functions. With DAQ, these functions have been separated out into modules that can be selected when invoking Snort. DAQ was created to integrate inline functionality and to provide flexibility for new modules.

The separation of DAQ also allows developers to create their own modules. There are already three externally developed DAQ modules – PF_RING, Napatech and PCAPRR (Sourcefire, Inc).

DAQ has two prerequisites: libpcap for packet capture and libdnet for other network functions.

Snort no longer uses libnet -- which hasn't been maintained or updated for many years (Roberts) -- for packet construction (Combs, 2010).

Four of the six DAQ modules allow Snort to operate inline and drop packets. Previously, to use this functionality you had to compile Snort with a patch (Metcalf & Julien). The "inline" term generally refers to the position of the sensor placed in between two devices or networks.

These are the DAQ modules included with Snort:

Pcap: the default mode, used for sniffer and IDS modes
Afpacket: inline on Linux using two bridged interfaces
Ipq: inline on Linux using netfilter, replaces the snort_inline project patch
Nfq: inline on Linux using netfilter
Ipfw: inline on OpenBSD and FreeBSD using divert sockets with the pf and ipfw firewalls
Dump: allows testing of inline and normalization mechanisms

Chris Murphy, chrismrph0@gmail.com

The key examples in this paper will discuss DAQ's inline functionality so it is worthwhile to first discuss the history of Snort as an IPS.

1.1. History

Snort began as a passive IDS only. Then in November 2000, Todd Lewis announced a proof of concept (Lewis, 2000) in which he modified Snort to use a new feature of the Linux netfilter firewall called QUEUE. The QUEUE target allowed the iptables configuration to request that netfilter switch packets out of kernel space and into a user space application for evaluation. In Lewis's example, Snort would be the user space application to receive and evaluate the packets. He modified his Snort rules so they would include a new action called FIREWALL to drop instead of alert on any packets that matched a specified pattern.

Over time, other Snort add-ons that offered inline capability--such as SnortSam (Knobbe) and fwsnort (Rash)--became available.

The option that would later be improved upon with the DAQ ipfw and nfq/ipq modules was created by the snort_inline open source project (Metcalf & Julien). The add-on was implemented as a patch to Snort that could be included during compilation. For legacy purposes, this original snort_inline functionality is now included with DAQ as the ipq module. The snort_inline patch used netfilter on Linux or ipfw on FreeBSD to drop traffic based on decisions made by Snort and its rules.

1.2. Snort as an IPS

Common advice for deploying Snort in an inline configuration cautions against false positives (Ierace, Urrutia, & Bassett, 2005). False positives describe a condition in which legitimate, non-harmful network traffic coincidentally matches a Snort rule and generates an alert. These false alerts can occur for numerous reasons such as when rules are not crafted precisely enough. False positives, already common with Snort in passive IDS mode, are amplified with Snort running in-line because now they have the potential to break legitimate network communications.

IPS false positives can be particularly dangerous on a large or complex network. Envision a large enterprise where the security team manages the IPS and the network operations team manages all other network infrastructure. Depending on where the inline sensors are deployed, users may report mysterious connectivity problems with common services due to false positives. The troubleshooting process is complicated because two separate groups with separate systems have to work in concert to find the root cause.

One option for minimizing this danger is to select only a few highly reliable rules to drop traffic. These drop rules will have been tested thoroughly and found to be extremely consistent in IDS alerting (Cox & Gerg, 2004, p. 134). Of course, using only a handful of rules means you are limiting the IPS to prevent a small number of highly critical exploits. Another option involves placement of the sensor. Instead of deploying the sensor where it can interrupt the most critical network traffic, such as inline between two core switches or between a core switch and a user access switch, you can place the sensor in a more targeted location. Consider placement between a web proxy server and the inside interface of the perimeter firewall and only using Snort's WEB rules. Or, place the sensor in front of a vulnerable application server that cannot be updated with patches due to unique business requirements.

Later, this paper presents test results from two inline deployment scenarios in which Snort is used selectively, positioning sensors at specific locations on a network to protect particular systems and applications.

2. DAQ Pcap Module Overview

DAQ pcap mode is the default mode that runs if you do not explicitly select another DAQ mode (Snort Team, Snort Users Manual 2.9.2, 2011). In pcap mode, Snort can run in the classic "sniffer" mode similar to that of the tcpdump utility, it can record packets to log files or it can run in IDS mode as a daemon.

Sniffer mode is not very useful on a busy network because the packet details will scroll across your console screen too quickly to read (Cox & Gerg, 2004, p. 42). One way to make the sniffer mode more valuable for troubleshooting or investigations is to use filters to focus precisely on

what you are looking for. Provided you are connected to a SPAN switch port or a network tap, instead of running "snort –v –i eth0", you could run "snort –v –i eth0 host 172.16.100.53" to see all traffic to and from a specific host. If you were concerned the host might be infected with a network worm and wanted to see what other hosts it was connecting to, you could run "snort –v –i eth0 src 172.16.100.53 and 'tcp[13] & 2!=0'" to display only SYN packets. (Miessler). A high number of SYN packets sent from a single host to a large number of hosts in a short period of time might indicate malicious behavior.

Since sniffer mode is not very helpful, Snort's pcap module also offers the classic packet logging functionality. By taking the above command and appending a logging directory argument (snort –v –i eth0 –l /var/log/snort) Snort will log packets to a binary capture file in the specified directory that can be read by tcpdump or Snort or preferably opened in Wireshark for more careful analysis.

Finally, running Snort in IDS mode with the –D switch forces it to start as a daemon.

3. DAQ Afpacket Module Overview

The DAQ afpacket module is available only on Linux. Afpacket leverages Snort rules and two network interfaces to drop suspicious traffic without having to rely on a separate, external firewall like netfilter (Snort Team, daq-0.6.2 README). Instead, Snort configures a network interface pair as a transparent bridge.

Consider these other important caveats for successful afpacket mode operation:

Obviously, since the Snort process is maintaining the bridge, the Snort daemon must be running if you expect network traffic to pass between the two network segments. If Snort crashes, traffic between the segments will cease. It makes sense to have some monitoring facility restart the Snort daemon in the event of a crash or at least alert you so you can respond.

Normally, if you want to configure a Linux system as a transparent bridge, you must configure it with separate bridging utilities. However, with DAQ afpacket, Snort manages the bridging on its own. You only need to configure the interfaces to be "up" before Snort starts. They do not need

to have an IP address assigned. Though you can do this on the command line with ifconfig (ifconfig eth1 up), it makes more sense to start them automatically for persistence.

If the bridging is to work successfully, the interfaces must be running in promiscuous mode so they can receive *all* Ethernet frames sent on the segment and not just those destined for their own physical addresses.

3.1. Configuration of test system

The following test scenario used several virtual machines running under Virtual Box 4.1:

Attacker: BackTrack 5 R2 Linux
Snort inline sensor: Snort 2.9.2.1 and DAQ 0.6.2 on CentOS 6.2 Linux
Target: Microsoft Windows Server 2003 Standard Edition Service Pack 1 (unpatched)

3.2. Steps for successful inline operation with DAQ afpacket mode (Lysemose, 2012):

Configure network interfaces correctly
Configure Snort to start with the correct DAQ options
Configure selected reliable Snort rules to drop instead of alert
Ensure Snort starts persistently after a reboot
Monitor Snort to ensure it remains running

3.3. Configure network interfaces correctly

In CentOS and similar Linux distributions, you typically configure persistent network options for each interface in these files - /etc/sysconfig/network-scripts/ifcfg-<if-name> - using this syntax:

```
DEVICE=eth1
BOOTPROTO=none
ONBOOT=yes
PROMISC=yes
```

Unfortunately, the *PROMISC=yes* directive no longer works in CentOS 6 (Hodgson, 2010). Instead, start the sensing interfaces in promiscuous mode using the /etc/rc.local startup script:

```
/sbin/ifconfig eth1 up promisc
/sbin/ifconfig eth2 up promisc
```

You can still configure the eth0 management interface in /etc/sysconfig/network-scripts/ifcfg-eth0 and allow it to be started and managed by the Network service.

In contrast to the nfq and ipfw DAQ modules, afpacket does not depend on IP routing. Therefore, you do not have to enable the kernel option to allow routing between network interfaces like you do with nfq. The Snort sensor is positioned on the network between two network segments with an interface connected to each and then transparently forwards traffic between its two interfaces so both network segments appear to be part of one broadcast domain.

Importantly, hosts on both connected segments require correct subnet mask assignments. With a Snort sensor positioned inline using afpacket mode, the subnet mask assignment has to change in order for the bridging to work properly. With typical IP routing, two adjacent class C networks, each with 256 addresses, might look like this:

```
Network 192.168.100.0/24    Mask 255.255.255.0
Network 192.168.101.0/24    Mask 255.255.255.0
```

With Snort afpacket bridging, the two networks listed above must be super netted into a larger network with 512 addresses:

```
Network 192.168.100.0/23    Mask 255.255.254.0
```

All hosts on both segments now need to configure their subnet mask value to be 255.255.254.0. Their network and broadcast addresses change as well -- 192.168.100.0 and 192.168.101.255.

3.4. Configure Snort to start with the correct DAQ options

In the snort.conf configuration file, make these changes relevant to afpacket and inline operation:

```
config daq: afpacket
config daq_mode: inline
```

Leave the local.rules file enabled but disable all other rules. Use local.rules to list all the reliable rules to use for dropping traffic. (You could also create a custom rules file like ips.rules or drop.rules as long as you remember to reference it in snort.conf.)

```
# site specific rules
include $RULE_PATH/local.rules
#include $RULE_PATH/attack-responses.rules
 [...lines omitted to save space…]
```

Finally, set the HOME_NET variable to the correct value to complement your rules and set a logging directory.

3.5. Configure selected reliable Snort rules to drop instead of alert

Selecting reliable Snort rules is a must for successful inline operation. As discussed earlier, false positives generated by an IDS create unwanted noise but false positives generated by an IPS can break networking!

In each of the Snort rules you use, you must change the action keyword from ALERT to DROP if you want Snort to block the traffic. The DROP action will also write to the alert log.

In preparation for the following inline test, a Nessus vulnerability scan of an unpatched Windows Server 2003 Service Pack 1 computer revealed several high risk vulnerabilities. A search for exploit code in the Exploit Database (http://www.exploit-db.com) located an exploit created by Debasis Mohanty (aka Tr0y/nopsled) for the vulnerability detailed by Microsoft in security bulletin MS08-067. The exploit worked. A search for existing Snort rules in the Sourcefire Vulnerability Research Team (VRT) rules set and the Emerging Threats (Emerging Threats) rules collection found only one – SID 14782 from the Sourcefire VRT rules – that fired reliably each time the exploit ran.

```
drop tcp $EXTERNAL_NET any -> $HOME_NET [135,139,445,593,1024:] (msg:"NETBIOS DCERPC
NCACN-IP-TCP srvsvc NetrpPathCanonicalize path canonicalization stack overflow attempt ";
```

Chris Murphy, chrismrph0@gmail.com

```
flow:established,to_server; dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188;
dce_opnum:31,32; dce_stub_data;
pcre:"/^(\x00\x00\x00\x00|.{4}(\x00\x00\x00\x00|.{12}))/sR"; byte_jump:4,-4,multiplier
2,relative,align,dce; pcre:"/\x00\.\x00\.\x00[\x2f\x5c]/R"; metadata:policy balanced-ips
drop, policy security-ips drop, service netbios-ssn; reference:cve,2008-4250;
reference:url,technet.microsoft.com/en-us/security/bulletin/MS08-067;
classtype:attempted-admin; sid:14782; rev:12;)
```

3.6. Ensure Snort starts persistently after a reboot

The /etc/default/snort configuration file included with the Snort RPM packages is not appropriate for an afpacket network interface configuration. You can start the Snort daemon upon boot using the /etc/rc.local script instead:

```
/usr/local/bin/snort -D -d -Q -c /etc/snort/etc/snort.conf -i eth1:eth2
```

The Snort command line options include:

-D – run in daemon mode
-d – inspect the application layer, not just the traffic headers
-Q – run Snort in inline mode (superfluous if 'config daq_mode: inline' is set in snort.conf)
-c – use the Snort configuration file at this location
-i – use the specified network interfaces to create the bridge

3.7. Monitor Snort to ensure it remains running

There are many monitoring tools that can monitor a daemon, service or process and take action based on specified conditions. Use one to notify you if Snort stops and possibly, restart it.

3.8. DAQ Afpacket Test Scenario

The following scenario presents one way to deploy a Snort sensor inline to block bad traffic:

At a medium-sized publicly traded company, the Windows server hosting one of the firm's critical financial applications is vulnerable to a software flaw currently being exploited by an active network worm. The Operations team has asked the Accounting group for a maintenance window during which to apply a vendor provided security update. However, the Accounting

group, citing month-end closing activities and reporting deadlines to the Securities and Exchange Commission, asks that the server not be modified or rebooted at this time.

Since the vulnerable server cannot be patched and the risk of infection is high, the security team instead decides to deploy a Snort sensor inline between the application server network and the user access network. Using the DAQ afpacket module, they enable only one NETBIOS rule to drop packets in the event the exploit used by the network worm is targeted against the server.

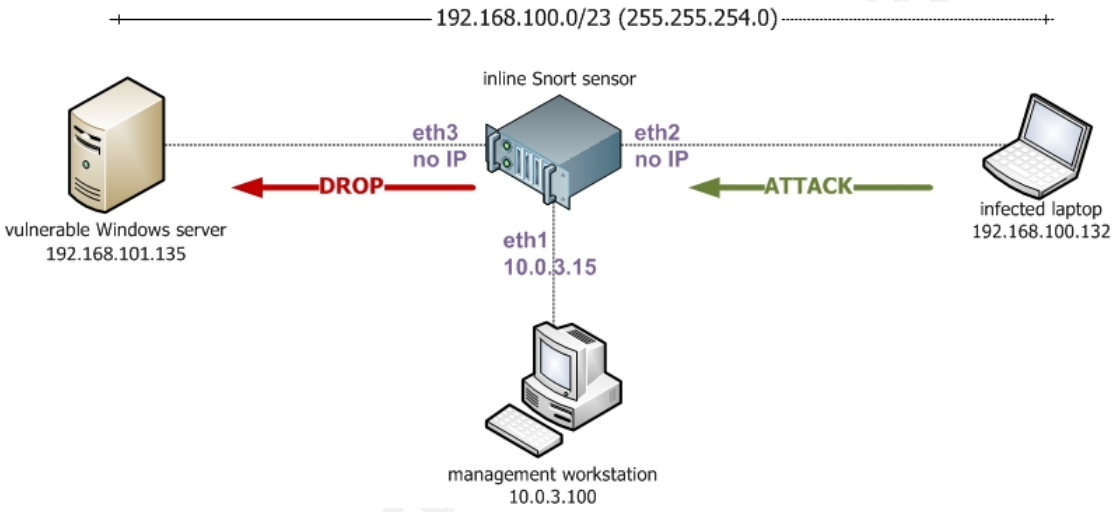

Figure 1 Inline Snort sensor using DAQ afpacket

3.9. Example

NOTE: Though manual steps illustrate the process here, the "network worm" in the scenario above would automate these steps with its own versions of the exploit and payload.

Snort was started to run in inline mode with the rule action set to alert. The NetrpPathCanonicalize exploit ran against the target machine:

```
./7132.py 192.168.101.135 2
```

7132.py is the Python exploit script written by Debasis Mohanty and the '2' argument selects a Windows 2003 payload.

Chris Murphy, chrismrph0@gmail.com

The exploit ran successfully and the output advised to telnet to the target computer on tcp port 4444. Netcat successfully connected to the target computer and displayed a command shell. The hostname command showed the computer name and the whoami command showed the session was running under the system security context.

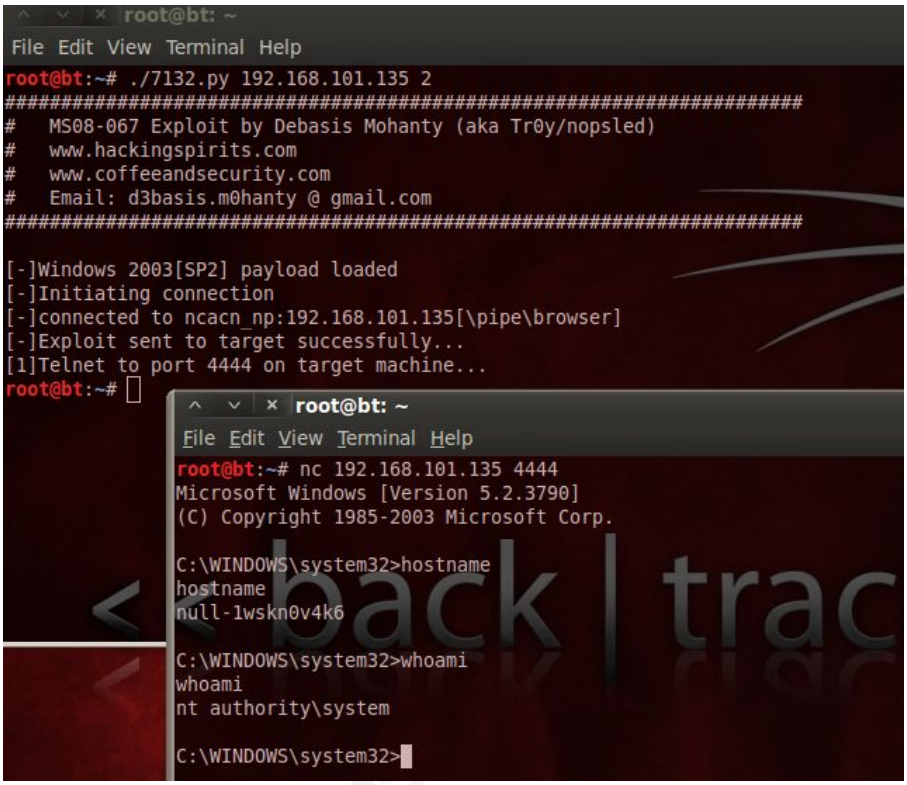

Figure 2 Exploit being used against a vulnerable Windows server

'netstat-an' output on the target machine showed the attacker's established session on tcp port 4444.

Chris Murphy, chrismrph0@gmail.com

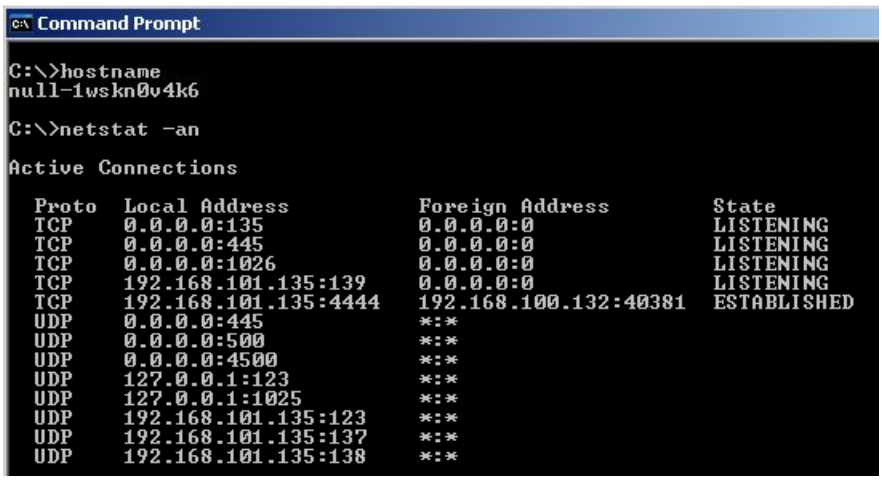

Figure 3 Netstat on the victim host showing the attacker's connection

The Snort alert file showed alerts that confirmed that Snort detected the exploit.

Next, the Snort process was stopped, the Windows target computer rebooted and the action changed on the Snort rule to drop instead of alert.

The exploit ran against the target machine again. The output looked the same as before and again reported success. However, this time netcat was not able to connect to the target machine.

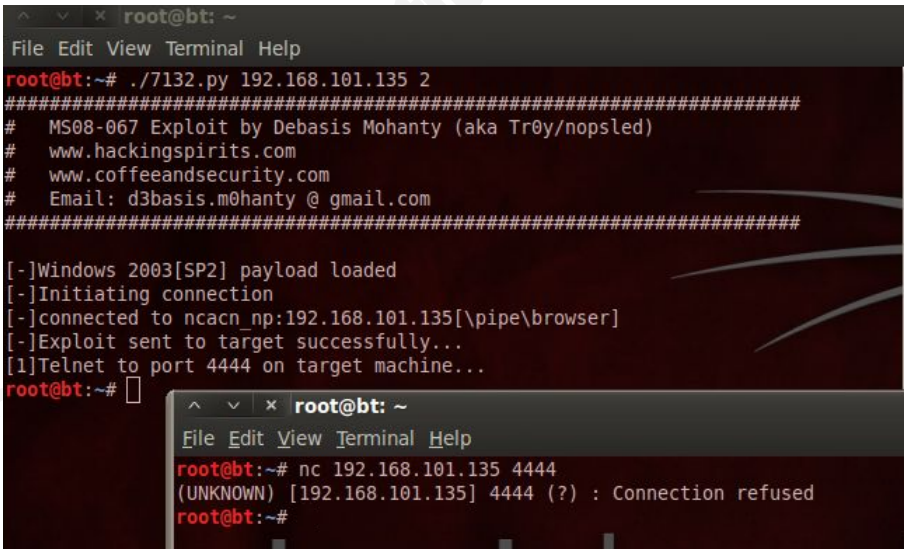

Figure 4 Exploit targeted at Windows server failed to work

Chris Murphy, chrismrph0@gmail.com

'netstat -an' on the target Windows computer showed no listener on tcp port 4444 and no active session from the attacking computer.

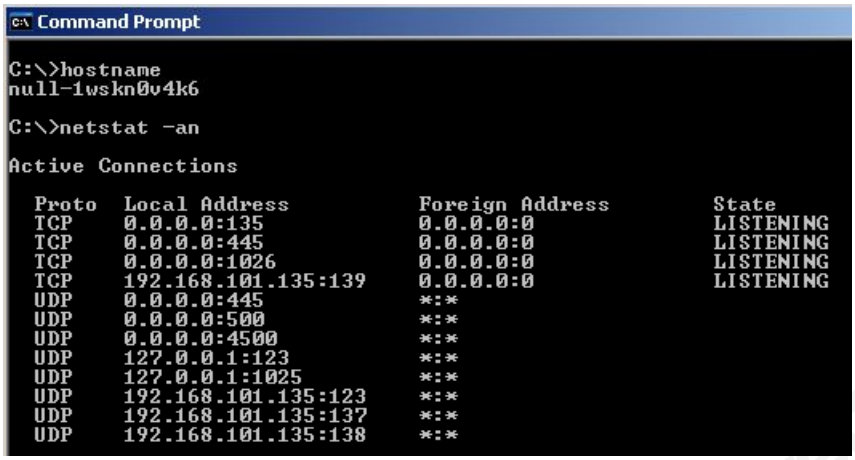

Figure 5 Netstat on the victim host showing no connection from the attacker

After stopping the Snort process, the reported statistics showed one alert and one corresponding blacklist verdict.

```
Action Stats:
     Alerts:              1 (  2.778%)
     Logged:              1 (  2.778%)
     Passed:              0 (  0.000%)
Limits:
      Match:              0
      Queue:              0
        Log:              0
      Event:              0
      Alert:              0
Verdicts:
      Allow:             35 ( 97.222%)
      Block:              0 (  0.000%)
    Replace:              0 (  0.000%)
  Whitelist:              0 (  0.000%)
  Blacklist:              1 (  2.778%)
     Ignore:              0 (  0.000%
```

The blacklist verdict represents Snort dropping the packet (Snort Team, README.counts).

Finally, the Snort alert log showed the details of the alert.

```
[**] [1:14782:12] NETBIOS DCERPC NCACN-IP-TCP srvsvc NetrpPathCanonicalize path
canonicalization stack overflow attempt  [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
07/28-21:35:15.447586 192.168.100.132:35462 -> 192.168.101.135:445
```

Chris Murphy, chrismrph0@gmail.com

```
TCP TTL:64 TOS:0x0 ID:25573 IpLen:20 DgmLen:734 DF
***AP*** Seq: 0xF6F2D249  Ack: 0xEFAA76D6  Win: 0x45A  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1880922 595
[Xref => http://technet.microsoft.com/en-us/security/bulletin/MS08-067][Xref =>
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2008-4250]
```

A simultaneous packet capture taken with tcpdump displayed the traffic on the eth1 interface. The packet that contains the attack is highlighted in red.

```
21:45:31.272919 IP 192.168.100.132.42164 > 192.168.101.135.microsoft-ds: Flags [P.], seq
202:298, ack 303, win 980, options [nop,nop,TS val 45653 ecr 4686], length 96SMB PACKET:
SMBntcreateX (REQUEST)

21:45:31.279493 IP 192.168.100.132.42164 > 192.168.101.135.microsoft-ds: Flags [P.], seq
298:448, ack 442, win 1047, options [nop,nop,TS val 45654 ecr 4687], length 150SMB
PACKET: SMBtrans (REQUEST)

21:45:31.281061 IP 192.168.100.132.42164 > 192.168.101.135.microsoft-ds: Flags [P.], seq
448:1130, ack 570, win 1114, options [nop,nop,TS val 45655 ecr 4687], length 682SMB
PACKET: SMBtrans (REQUEST)

21:45:31.286014 IP 192.168.100.132.42164 > 192.168.101.135.microsoft-ds: Flags [F.], seq
1130, ack 570, win 1114, options [nop,nop,TS val 45656 ecr 4687], length 0
```

Wireshark decoded the packet and thus shows more detail. Notice the decoded NetPathCanonicalize request.

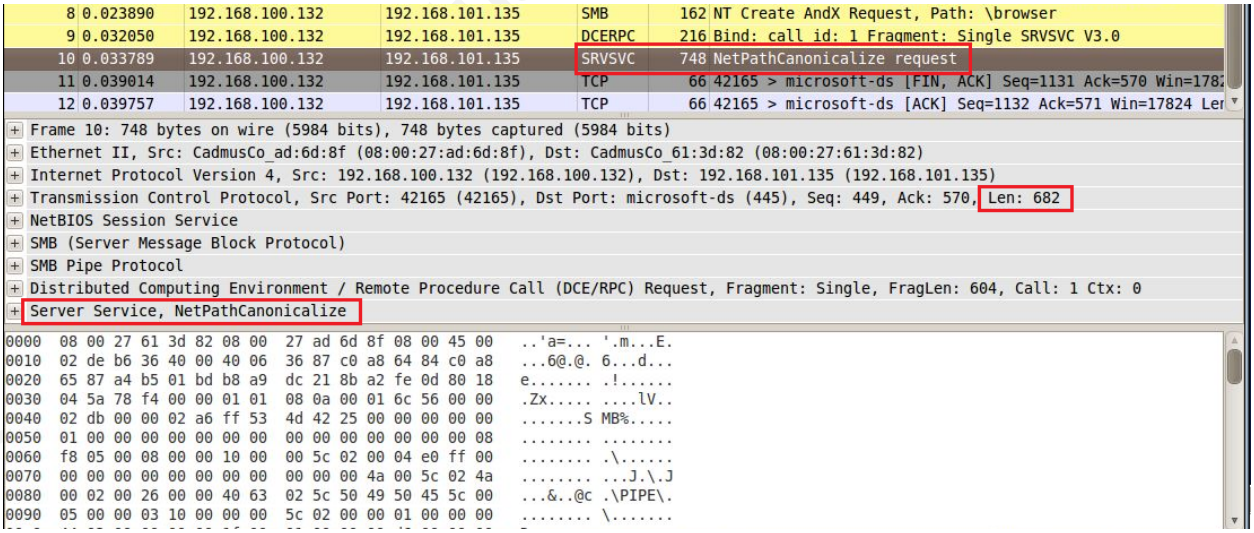

Figure 6 Attack decoded by Wireshark

Chris Murphy, chrismrph0@gmail.com

Another packet capture recorded with tcpdump showed the traffic on eth2 that was forwarded by Snort over the afpacket bridge. Notice the packet showing "length 682SMB PACKET: SMBtrans (REQUEST)" is not present because it was dropped by Snort.

```
21:45:31.273098 IP 192.168.100.132.42164 > 192.168.101.135.microsoft-ds: Flags [P.], seq
202:298, ack 303, win 980, options [nop,nop,TS val 45653 ecr 4686], length 96SMB PACKET:
SMBntcreateX (REQUEST)

21:45:31.279633 IP 192.168.100.132.42164 > 192.168.101.135.microsoft-ds: Flags [P.], seq
298:448, ack 442, win 1047, options [nop,nop,TS val 45654 ecr 4687], length 150SMB
PACKET: SMBtrans (REQUEST)

21:45:31.291909 IP 192.168.100.132.42164 > 192.168.101.135.microsoft-ds: Flags [R.], seq
448, ack 570, win 0, length
```

4. DAQ Ipq Module Overview

The ipq module offers the same functionality provided by the snort_inline patch in earlier versions of Snort. It leverages the QUEUE target in netfilter to move packets from the kernel to the user space application – Snort, in this case – for evaluation.

This mode has been included with DAQ to support those with legacy configurations who used the snort_inline patch in previous Snort versions. (Snort Team, daq-0.6.2 README) (Combs, 2010). The new DAQ module to integrate netfilter and Snort is called nfq and is covered in the next section.

5. DAQ Nfq Module Overview

The nfq module leverages the QUEUE target in netfilter to move packets from the kernel to a user space application for evaluation.

5.1. Details of test system

The following test scenario used several virtual machines running under Virtual Box 4.1:

Attacker: BackTrack 5 R2 Linux
Snort inline sensor: Snort 2.9.2.1 and DAQ 0.6.2 on Debian GNU/Linux 6.0.4 (squeeze)
Target: Open Web Application Security Project (OWASP) Broken Web Applications VM Version 0.94

Chris Murphy, chrismrph0@gmail.com

5.2. Steps for successful inline operation with DAQ nfq mode (Snort Team, daq-0.6.2 README):

Enable IP forwarding in the kernel
Configure interfaces correctly
Configure Snort to start with the correct DAQ options
Configure select group of reliable Snort rules to drop instead of alert
Configure iptables to appropriately filter traffic according to your needs
Configure iptables to appropriately use Network Address Translation (NAT) and forward according to your needs (optional)
Configure iptables to queue traffic according to your needs
Ensure Snort starts persistently after a reboot
Monitor Snort to ensure it remains running

5.3. Enable IP forwarding in the kernel

You must enable routing so netfilter can route packets between the external and internal network interfaces. You can use the sysctl command to enable this routing until the next reboot, but in order for the support to be persistent, you should enable it in /etc/sysctl.conf:

```
net.ipv4.ip_forward=1
```

5.4. Configure interfaces correctly

On a Debian Linux system, configure the network interfaces like this in the /etc/network/interfaces file:

```
# management interface
auto eth0
iface eth0 inet dhcp
# external facing interface
auto eth1
iface eth1 inet static
```

Chris Murphy, chrismrph0@gmail.com

```
        address 192.168.100.145
        netmask 255.255.255.0
# internal facing interface
auto eth2
iface eth2 inet static
        address 192.168.101.145
        netmask 255.255.255.0
```

5.5. Configure Snort to start with the correct DAQ options

In snort.conf, set the HOME_NET variable to the IP address assigned to the external interface of the gateway server -- ipvar HOME_NET [192.168.100.145/32] – and configure the log directory -- config logdir: /var/log/snort. Comment out all rule files except the local rules -- include $RULE_PATH/local.rules. The local.rules file will hold all the reliable Snort rules that you are comfortable configuring to drop. Recall from the previous discussion of Snort's inline functionality that the default rule set is not appropriate for an inline sensor given the risk of false positives and interrupted network communication.

These snort.conf options configure nfq. You do not actually need the last line since the default queue is already zero. You can use this rule in the iptables filter table -- "-A FORWARD -j NFQUEUE". However, you must specify it here if you choose a different queue number in your policy.

```
config daq: nfq
config daq_mode: inline
config daq_var: queue=0
```

Snort developers recommend setting the snap length to the highest possible value since fragmented packets are defragmented before being moved to the QUEUE (Snort Team, daq-0.6.2 README):

```
config snaplen: 65535
```

With this maximum value, Snort will be sure to see the entire packet.

5.6. Configure select group of reliable Snort rules to drop instead of alert

Testing identified a Snort rule from the Emerging Threats collection that consistently alerted for an exploit written by Kingcope (kingcope, 2011) to trigger a denial of service vulnerability on an unpatched Apache web server:

```
drop tcp any any -> any $HTTP_PORTS (msg:"ET SCAN Kingcope KillApache.pl Apache
mod_deflate DoS attempt"; flow:established,to_server; content:"Range|3a|bytes=0-,5-0,5-
1,5-2,5-3,5-4,5-5,5-6,5-7,5-8,5-9,5-10,5-11,5-12,5-13,5-14"; http_header;
fast_pattern:only;reference:url,seclists.org/fulldisclosure/2011/Aug/175;
classtype:attempted-dos; sid:2013472; rev:2;)
```

5.7. Configure iptables to appropriately filter traffic according to your needs

Choose a rule set that protects the host from the external (Internet) network. At a minimum, allow packets from already established sessions to pass into the firewall. Allow ssh requests from the trusted management network. You must allow http requests from anywhere if you are providing web services. Finally, block all other requests.

5.8. Configure iptables to appropriately use Network Address Translation (NAT) and forward according to your needs

NAT is optional for DAQ nfq. You will only need it if the Snort sensor is functioning as a gateway and protecting hosts on a separate network. If you are using the Snort sensor to protect services running on itself, you do not need to configure NAT.

5.9. Configure iptables to queue traffic according to your needs.

The QUEUE target number must match that specified in the Snort DAQ configuration if you override the default.

```
*filter
:FORWARD ACCEPT [0:0]
-A FORWARD -j NFQUEUE
COMMIT
```

Chris Murphy, chrismrph0@gmail.com

This simple rule queues incoming packets so Snort can evaluate them (Snort Team, daq-0.6.2 README).

5.10. Ensure Snort starts persistently after a reboot

You can use /etc/rc.local to start Snort automatically upon boot:

```
/usr/local/bin/snort -d -D -c /etc/snort/etc/snort.conf
```

If you need to confirm, the Snort messages will be recorded in /var/log/daemon.log after the system boots:

```
Aug  1 23:53:10 debian2 snort[1907]: nfq DAQ configured to inline.
Aug  1 23:53:10 debian2 snort[1907]: The DAQ version does not support reload.
Aug  1 23:53:10 debian2 snort[1907]: Initializing daemon mode
Aug  1 23:53:10 debian2 snort[1919]: Daemon initialized, signaled parent pid: 19
07
```

"Nfq DAQ configured to inline" confirms that nfq is ready to drop traffic according to your rule set.

5.11. Monitor Snort to ensure it remains running

If Snort is not running, netfilter will still queue packets and wait for evaluation. Network communication through the firewall will cease since there will be no user space application to respond. There are many monitoring tools that can monitor a daemon, service or process and take action based on specified conditions. You should configure one of these services to re-start Snort or at least alert you so you can investigate.

5.12. DAQ Nfq Test Scenario

The following scenario presents one way to deploy a Snort sensor inline to block bad traffic using netfilter:

At a small trade association, a web server has been installed in a DMZ network. The web server is protected from the Internet by a Linux gateway server running the netfilter firewall. However, due to its role as a public web server, it must accept http requests from any address on the Internet. Details emerge of an easy to launch and highly effective denial of service attack

Chris Murphy, chrismrph0@gmail.com

against the Apache web server. There is currently no patch available. A consultant working with the association's IT staff recommends installing Snort on the Linux gateway and configuring the DAQ nfq module. In the Snort rules configuration, only the local rules are enabled. A specific rule to detect the DoS is added with the default action changed from ALERT to DROP. The iptables configuration will redirect all incoming tcp port 80 traffic to the NFQUEUE target so it can be inspected by Snort, which will render a verdict whether to allow or block.

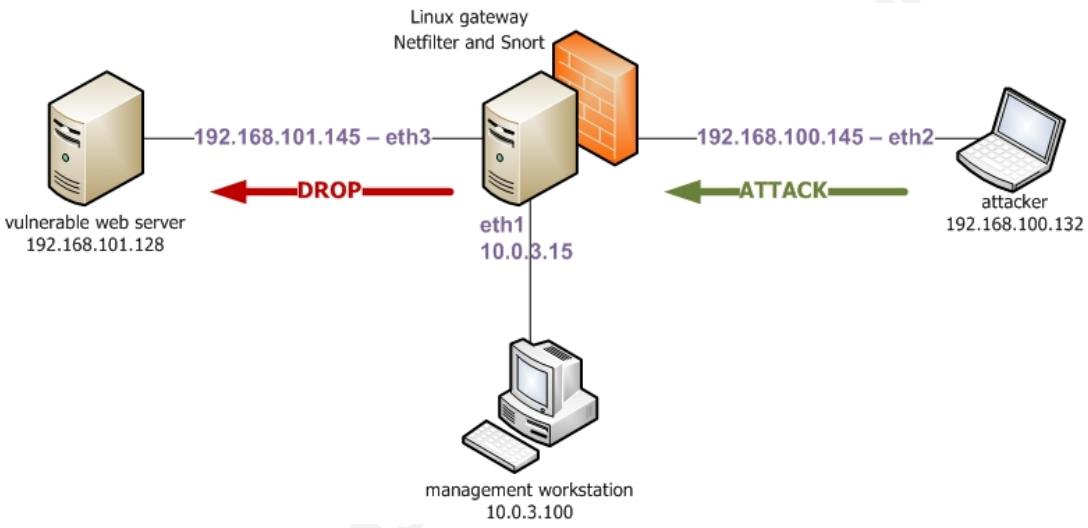

Figure 7 Inline Snort sensor using DAQ nfq

5.13. Example

The killapache.pl exploit ran against the target web server and specified the recommended 50 "forks":

```
root@bt:~# perl ./killapache.pl 192.168.100.145 50
host seems vuln
ATTACKING 192.168.100.145 [using 50 forks]
:pPpPpppPpPPppPpppPp
ATTACKING 192.168.100.145 [using 50 forks]
```

[...lines omitted to save space…]

Chris Murphy, chrismrph0@gmail.com

After a few moments, the exploit began to work as “out of memory” errors were written to the web server’s console:

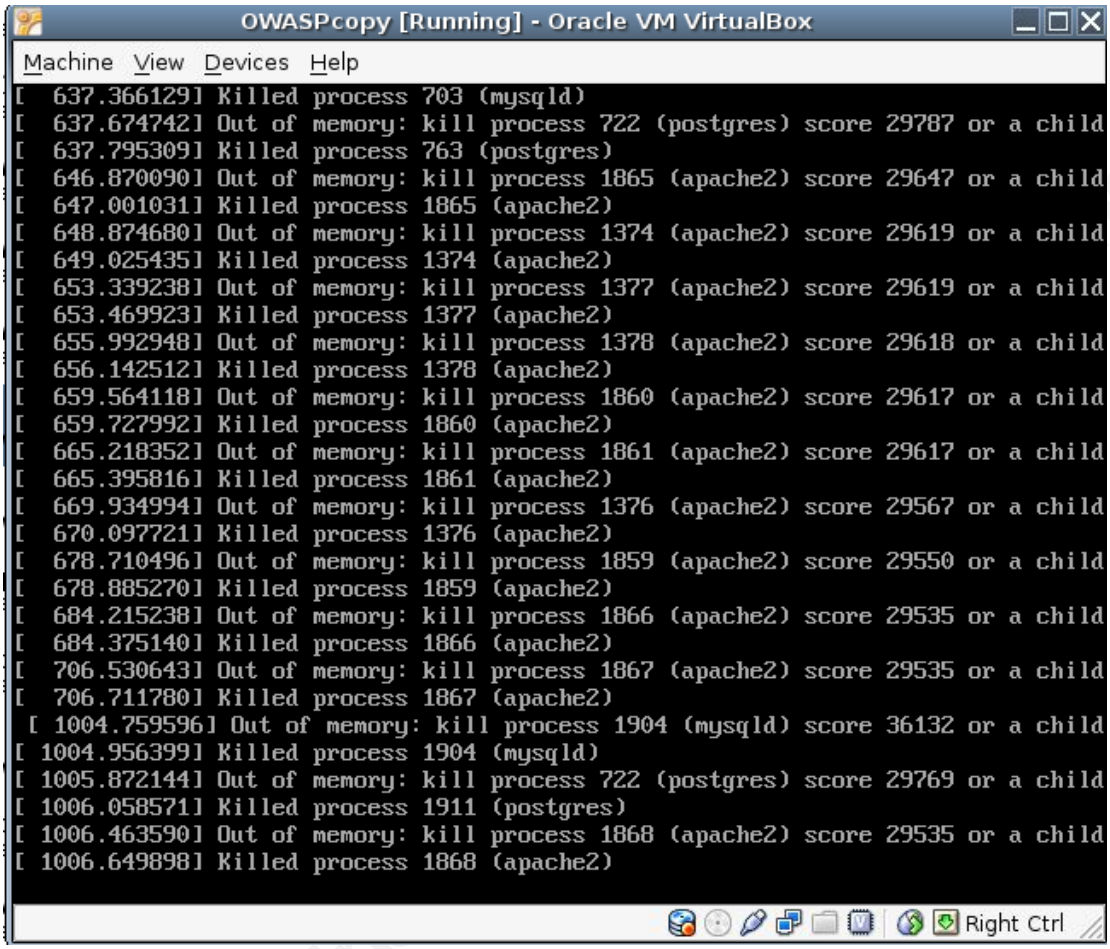

Figure 8 Apache web server under denial of service attack

The web server returned an error when connecting to an URL with a browser:

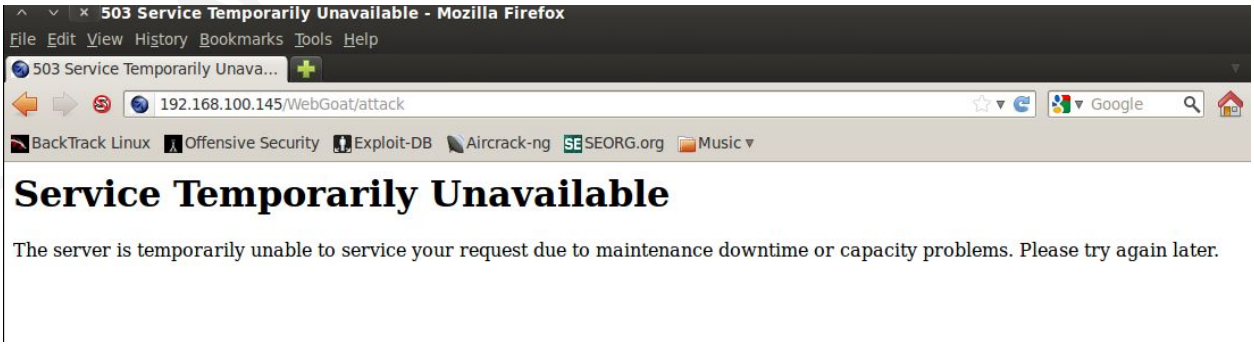

Figure 9 Apache web server not available due to denial of service attack

The output of the ‘top’ command on the target web server showed an extremely high load average – that is, the processor utilization for the last 1, 5 and 15 minute periods. On a single

processor system like the test host, any whole number value means the CPU is overloaded. (Walker, 2006)

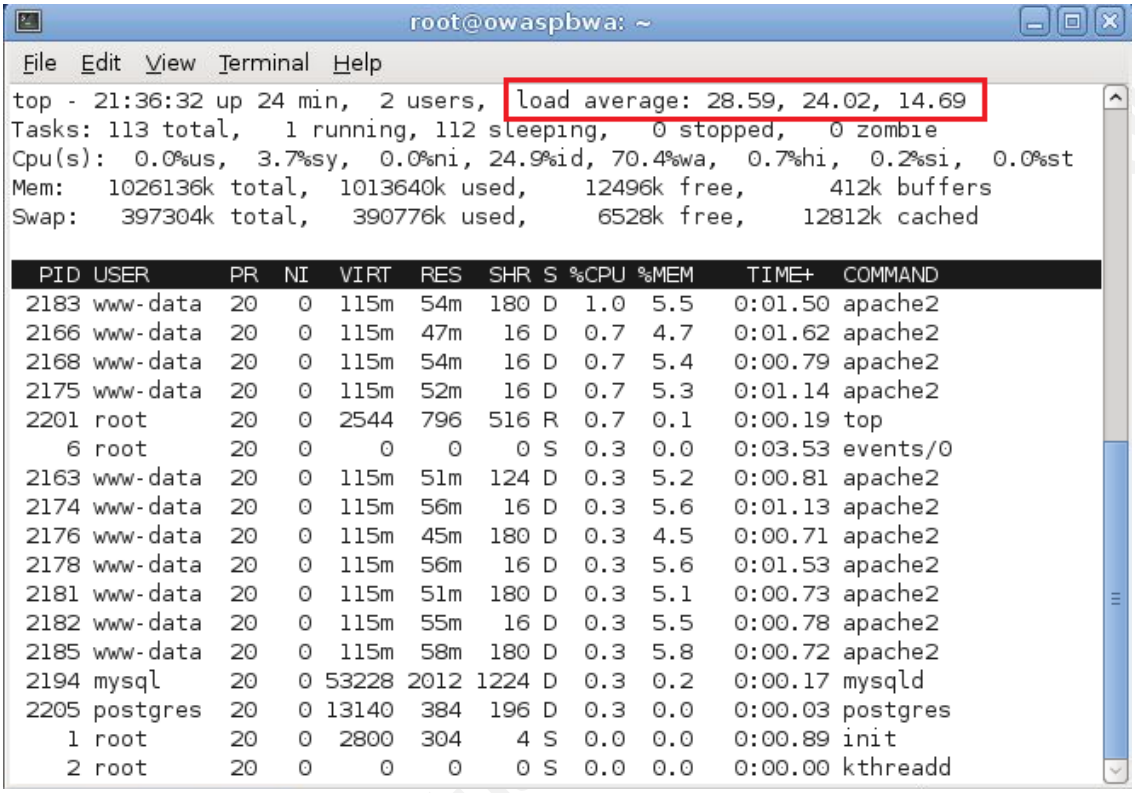

Figure 10 High load average caused by denial of service attack

The Snort alert log showed multiple alerts related to the exploit:

```
[**] [1:2013472:2] ET SCAN Kingcope KillApache.pl Apache mod_deflate DoS attempt [**]
[Classification: Attempted Denial of Service] [Priority: 2]
08/01-21:33:02.985828 192.168.100.132:57591 -> 192.168.101.128:80
TCP TTL:63 TOS:0x0 ID:32551 IpLen:20 DgmLen:8142 DF
***AP*** Seq: 0x77A0DA27  Ack: 0xE6C21BDF  Win: 0x38C0  TcpLen: 32
[Xref => http://seclists.org/fulldisclosure/2011/Aug/175]
```

After stopping Snort and resetting the target web server, the action in the Snort rule was changed from alert to drop. The exploit re-ran but this time the web server remained accessible and the load average remained normal.

Chris Murphy, chrismrph0@gmail.com

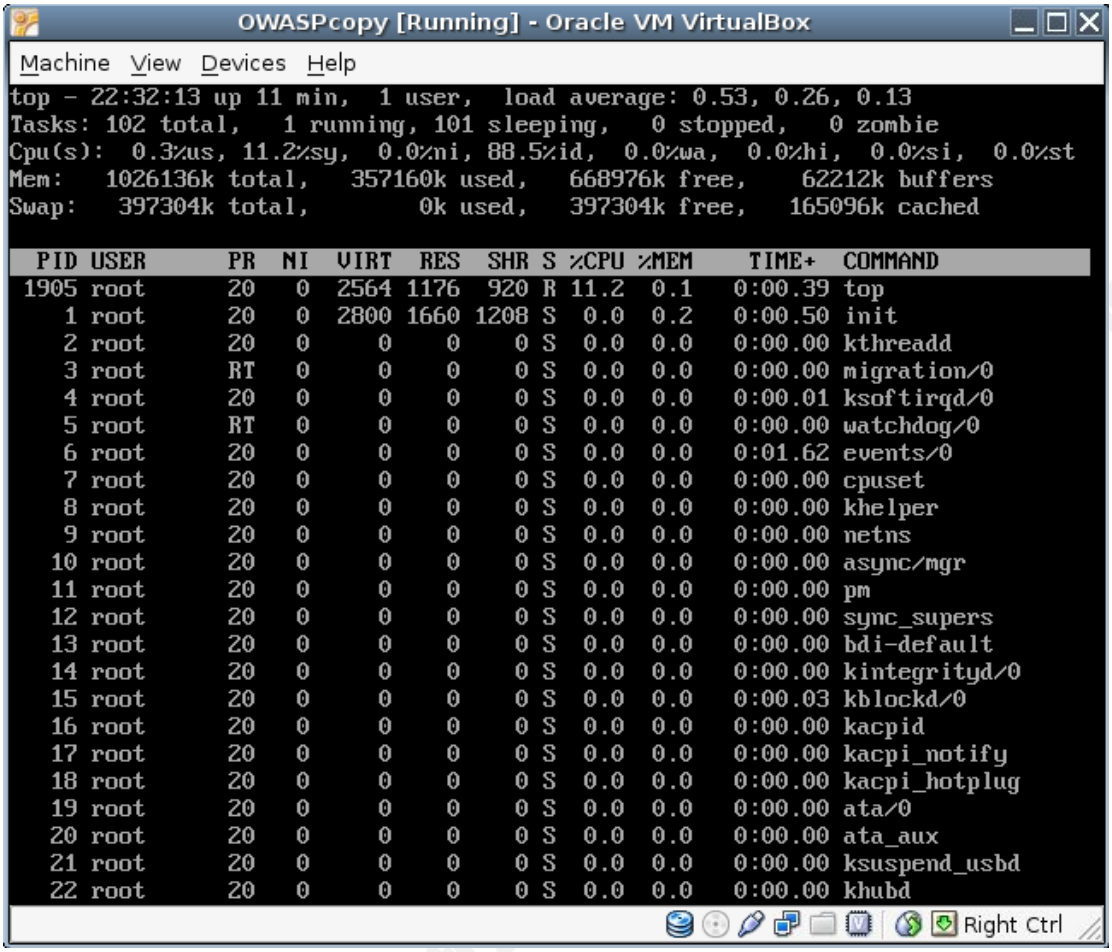

Figure 11 Normal load average on targeted web server

When Snort stopped, the statistics reported 50 alerts and 50 blacklist verdicts, corresponding to the 50 forks specified as an argument to the killapache.pl exploit:

```
Action Stats:
     Alerts:            50 (  2.509%)
     Logged:            50 (  2.509%)
     Passed:             0 (  0.000%)
Limits:
      Match:             0
      Queue:             0
        Log:             0
      Event:             0
      Alert:             0
Verdicts:
      Allow:           743 ( 37.280%)
      Block:          1200 ( 60.211%)
     Replace:            0 (  0.000%)
  Whitelist:             0 (  0.000%)
  Blacklist:            50 (  2.509%)
     Ignore:             0 (  0.000%)
```

Chris Murphy, chrismrph0@gmail.com

While the attack was running, all http packets on the eth1 and eth2 interfaces were captured with tcpdump. The Tshark utility (related to Wireshark) provided summary statistics:

Output from "tshark -z http,tree -r eth1-apache-attack.pcap":

```
================================================================
 HTTP/Packet Counter            value          rate         percent
----------------------------------------------------------------
 Total HTTP Packets                61        0.000109
  HTTP Request Packets             61        0.000109          100.00%
   HEAD                            51        0.000091           83.61%
   GET                             10        0.000018           16.39%
```

Output from "tshark -z http,tree -r eth2-apache-attack.pcap":

```
================================================================
 HTTP/Packet Counter            value          rate         percent
----------------------------------------------------------------
 Total HTTP Packets                11        0.000020
  HTTP Request Packets             11        0.000020          100.00%
   HEAD                             1        0.000002            9.09%
   GET                             10        0.000018           90.91%
```

The killapache.pl exploit sent 50 packets. With Snort running in inline mode, 61 packets entered the eth1 interface from the "Internet" side but only 11 were redirected to the internal web server by netfilter. The 11 requests were legitimate requests made with a web browser. 50 packets were successfully dropped by Snort's DAQ nfq module.

6. DAQ Ipfw Module Overview

IPFW mode leverages the divert sockets functionality supported in FreeBSD and OpenBSD.

Divert sockets is similar to the netfilter QUEUE target because it allows the pf or ipfw firewalls running in the kernel to pass traffic off to a user space application for evaluation before inserting it back into the kernel to continue normal processing.

The following simple example shows the pf firewall on OpenBSD using divert sockets. It also shows Snort seeing the diverted packets.

6.1. OpenBSD and pf

These pf rules (excerpted from a larger ruleset) running on an OpenBSD gateway server allow http requests in and redirect them to an internal web server. They also divert traffic to port 5317 (Hansteen, 2011) (Teo, 2011) (OpenBSD developers, 2012).

> # pfctl –s rules
> pass out on pcn0:network all flags S/SA keep state scrub (reassemble tcp) divert-packet port 5317 nat-to (pcn0:0)
> pass in on pcn0 inet proto tcp from any to (pcn0:0) port = www flags S/SA keep state scrub (reassemble tcp) divert-packet port 5317 rdr-to 192.168.101.128 port 80

The following Snort command invokes the DAQ ipfw in inline mode and specifies port 5317 as the diverted port. Snort is running in verbose mode and prints captured packets to the console screen.

```
# snort --daq-dir /usr/local/lib/daq -v -Q -daq ipfw --daq-var port=5317 -c
/etc/snort/etc/snort.conf
Enabling inline operation
Running in IDS mode

        --== Initializing Snort ==--
```

[…lines omitted to save space…]

```
ipfw DAQ configured to inline.
The DAQ version does not support reload.
Reload thread starting...
Reload thread started, thread 0x86ef4c00 (14434)

        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.2.2 IPv6 GRE (Build 121)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/snort/snort-team
```

[…lines omitted to save space…]

```
Commencing packet processing (pid=14434)
Decoding Raw IP4
08/08-19:54:44.743745 192.168.100.132:49448 -> 192.168.101.128:80
TCP TTL:64 TOS:0x0 ID:56486 IpLen:20 DgmLen:60 DF
******S* Seq: 0x45575FCF  Ack: 0x0  Win: 0x3908  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 2382744701 0 NOP WS: 4
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

```
08/08-19:54:44.748903 192.168.100.129:80 -> 192.168.100.132:49448
TCP TTL:63 TOS:0x0 ID:0 IpLen:20 DgmLen:60 DF
***A**S* Seq: 0x15E6D3E4  Ack: 0x45575FD0  Win: 0x16A0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 3648608240 33641 NOP
TCP Options => WS: 6
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

[…lines omitted to save space…]

7. DAQ Dump Module Overview

The DAQ dump module allows you to test inline normalization functions (Snort Team, README.normalize).

The Snort normalize pre-processors can edit incoming IP and TCP packets to remove non-compliant or less often used options to reduce the risk of IDS/IPS evasion. Evasion is a common danger for intrusion detection (Ptacek & Newsham, 1998). When an IDS evaluates traffic ultimately destined for another host on the network, it must see it and process the traffic the same way that other machine would if it is to effectively detect or block an attack. If it does not, it could miss the attack, producing a false negative error. If the IDS is viewing traffic through the TCP/IP "lens" of a Windows computer, but the target is a Cisco router, the attack could go undetected due to slight differences in their TCP/IP implementations. The Snort normalize pre-processors remove any unusual or rarely used IP or TCP options to increase the effectiveness of the IDS. For example, the normalize_ip4 pre-processor can remove the Type Of Service (TOS) bit. A detailed example follows below. The normalize_tcp pre-processor can remove any data included in SYN or RST packets which would clearly be anomalous. The pre-processors support many more options as detailed in the README.normalize document and the Snort manual.

Testing normalization would ordinarily require a working inline sensor and configuration. That means at the very least, you must have a physical or virtual computer with two interfaces connected to two different network segments. Even then, in order to test, you would have to run Snort in interactive mode, pass some traffic, and then press Ctrl + C to kill the Snort process. Only then will you be able to see the results of your testing in the Snort statistics reported to the console and the pcap log file.

Chris Murphy, chrismrph0@gmail.com

The dump module allows you to test the same scenarios without having the physical system and configuration in place. It also creates a helpful inline-out.pcap file containing the packet traces from testing.

7.1. Example

The hping3 utility sent packets to a target system and specified an arbitrary type of service (TOS) hex value of 0x8 requesting "maximum throughput".

```
root@bt:~# hping3 --tos 08 192.168.100.200
HPING 192.168.100.200 (eth1 192.168.100.200): NO FLAGS are set, 40 headers + 0 data bytes
len=46 ip=192.168.100.200 ttl=64 DF id=0 sport=0 flags=RA seq=0 win=0 rtt=1.7 ms
len=46 ip=192.168.100.200 ttl=64 DF id=0 sport=0 flags=RA seq=1 win=0 rtt=0.8 ms
```

These packets were recorded into a tos.pcap capture file with tcpump. In snort.conf, all the normalize preprocessors were commented out and disabled. The DAQ dump read the tos.pcap file and wrote the before-normal.pcap file:

```
snort -r tos.pcap -Q --daq dump --daq-var file=before-normal.pcap --daq-var load-
mode=read-file -c /etc/snort/etc/snort.conf
```

This is the first packet captured in the before-normal.pcap file. Notice that the TOS bit was set to 08:

```
08/11-13:58:26.841569 192.168.100.132:2479 -> 192.168.100.200:0
TCP TTL:64 TOS:0x8 ID:17115 IpLen:20 DgmLen:40
******** Seq: 0x21E49478  Ack: 0x7E20DF9E  Win: 0x200  TcpLen: 20
```

In snort.conf, "tos" was added to the normalize_ip4 configuration and the option was uncommented to enable it.

```
preprocessor normalize_ip4: tos
```

DAQ dump ran again to read the same tos.pcap file:

```
snort -r tos.pcap -Q --daq dump --daq-var file=after-normal.pcap --daq-var load-
mode=read-file -c /etc/snort/etc/snort.conf
```

Chris Murphy, chrismrph0@gmail.com

This time, the first packet in the after-normal.pcap file showed the TOS bit was now zero:

```
08/11-13:58:26.841569 192.168.100.132:2479 -> 192.168.100.200:0
TCP TTL:64 TOS:0x0 ID:17115 IpLen:20 DgmLen:40
******** Seq: 0x21E49478  Ack: 0x7E20DF9E  Win: 0x200  TcpLen: 20
```

When Snort completed reading the file, it stopped and displayed statistics. Under Action Stats, it showed 48 "replaced" verdicts. That number corresponded to the ip4::tos number under Normalizer statistics.

```
Action Stats:
     Alerts:            0 (  0.000%)
     Logged:            0 (  0.000%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:            0
Verdicts:
      Allow:            4 (  7.692%)
      Block:            0 (  0.000%)
    Replace:           48 ( 92.308%)
  Whitelist:            0 (  0.000%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
===============================================================================
Normalizer statistics:
              ip4::trim: 0
               ip4::tos: 48
```

The dump mode test showed that the Snort pre-processor would have reset the TOS bit back to zero no matter what it was originally set to.

8. Conclusion

The DAQ libraries were developed for Snort to provide flexibility; now Snort users can select different capturing modes from the command line or through the Snort configuration file according to their needs. The plug-in characteristics of the DAQ modules also allow developers outside the Snort project to develop their own libraries, ultimately making Snort more powerful. Finally, for those needing to use Snort in true intrusion prevention mode, several integrated choices exist so users do not have to seek out add-ons or patches.

Chris Murphy, chrismrph0@gmail.com

9. References

Bies, L. (n.d.). *Firewall configuration for iptables*. Retrieved from Lammert Bies: http://www.lammertbies.nl/comm/info/iptables.html

Combs, R. (2010, August 12). *VRT: Snort 2.9 Essentials: The DAQ*. Retrieved from VRT Blog: http://vrt-blog.snort.org/2010/08/snort-29-essentials-daq.html

(2004). In K. Cox, & C. Gerg, *Managing Security with Snort and IDS Tools.* Sebastopol: O'Reilly Media, Inc.

Emerging Threats. (n.d.). *emerging-all.rules.* Retrieved from Emerging Threats: http://rules.emergingthreats.net/open/snort-2.9.0/emerging-all.rules

FreeBSD developers. (n.d.). *IPFW.* Retrieved from FreeBSD Handbook: http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls-ipfw.html

Hansteen, P. N. (2011). *The Book of PF: a No-Nonsense Guide to the OpenBSD Firewall.* San Francisco: No Starch Press, Inc.

Hertzog, R., & Mas, R. (2012). *The Debian Administrator's Handbook.* Freexian SARL.

Hodgson, K. (2010, October 9). Grokbase.com [CentOS discussion group] RE: Software bridge setup in RHEL 5/CentOS 5 questions, possible bug.

Ierace, N., Urrutia, C., & Bassett, R. (2005, June). Intrusion Prevention Systems. *Ubiquity*.

kingcope. (2011, 8 19). *Apache httpd Remote Denial of Service (memory exhaustion).* Retrieved from Exploit Database: http://www.exploit-db.com/exploits/17696/

Knobbe, F. (n.d.). *SnortSam - A Firewall Blocking Agent for Snort*. Retrieved from About SnortSam: http://www.snortsam.net/

Lewis, T. (2000, 11 29). [Snort-users mailing list] announcement & questions: user space firewall.

Linux Home Networking. (n.d.). *Quick HOWTO : Ch14 : Linux Firewalls Using iptables.* Retrieved from Linux Home Networking: http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch14_:_Linux_Firewalls_Using_iptables

Linuxtopia. (n.d.). *DNAT Target.* Retrieved from Linux Packet Filtering and iptables: http://www.linuxtopia.org/Linux_Firewall_iptables/x4013.html

Lysemose, H. (2012, February 9). [Snort-users mailing list] Re: Basics of setting up an inline snort installation.

Chris Murphy, chrismrph0@gmail.com

Metcalf, W., & Julien, V. (n.d.). *snort_inline*. Retrieved from snort_inline: http://snort-inline.sourceforge.net/oldhome.html

Miessler, D. (n.d.). *A tcpdump Tutorial and Primer*. Retrieved from danielmiessler.com grep understanding: http://danielmiessler.com/study/tcpdump/

OpenBSD developers. (2012, March 21). *PF: Redirection (Port Forwarding).* Retrieved from OpenBSD: http://www.openbsd.org/faq/pf/rdr.html

Ptacek, T. H., & Newsham, T. N. (1998). *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection.* Secure Networks, Inc.

Rash, M. (n.d.). *fwsnort - iptables Intrusion Detection with String Matching and Snort Rules*. Retrieved from CipherDyne: http://cipherdyne.org/fwsnort/

Roberts, S. (n.d.). *sam-github/libnet*. Retrieved from GitHub, Inc. [US]: https://github.com/sam-github/libnet#readme

Rogness, N. (n.d.). *FreeBSD Firewall.* Retrieved from A Network Tutorial/How-To Guide for the FreeBSD OS: http://freebsd.rogness.net/redirect.cgi?basic/firewall.html

Rogness, N. (n.d.). *Nat and IPFW.* Retrieved from A Network Tutorial/How-To Guide for the FreeBSD OS: http://freebsd.rogness.net/redirect.cgi?basic/nat.html

Snort Team. (2011, December 7). *Snort Users Manual 2.9.2.* Retrieved from Snort: http://www.snort.org/assets/166/snort_manual.pdf

Snort Team. (n.d.). daq-0.6.2 README. Sourcefire, Inc.

Snort Team. (n.d.). README.counts. Sourcefire, Inc.

Snort Team. (n.d.). README.normalize. Sourcefire, Inc.

Sourcefire, Inc. (n.d.). *External DAQ Modules*. Retrieved from www.snort.org: http://www.snort.org/snort-downloads/external-daq/

Sourcefire, Inc. (n.d.). *Snort :: Home Page*. Retrieved from Snort: http://www.snort.org

Steve. (2005, January 7). *Port forwarding for iptables (DMZ).* Retrieved from Debian Administration: http://www.debian-administration.org/articles/73

Teo, L. (2011, October 4). [Old Nabble | openbsd dev-bugs] PF divert-packet with nat-to/rdr-to works in 4.9, breaks in -current.

voodoo. (2011, September 30). *IPtables NAT/SNAT/transparent proxy redirect rule examples*. Retrieved from Xinotes: http://www.xinotes.org/notes/note/1527/

Walker, R. (2006). Examining Load Average. *Linux Journal*, http://www.linuxjournal.com/article/9001.

Chris Murphy, chrismrph0@gmail.com