Global Information Assurance Certification Paper

University Security Audit

GCIA

Practical Assignment

[4.0]

Ian Marks
GCIA
Crystal City
Sept. 26,2004

Part 1: Executive Summary

The goal of this audit is to determine the state of the University's network security infrastructure. By defining the network boundaries and analyzing the outside traffic flow we will be able set a course of action to secure your Internet point of presence.

The first step is to identify your valued assets. In a University network you can categorize the assets in groups by:

- Technology resources (hardware and software)
- Information resources (grades, health records, payroll records, and personally identifiable information)
- Curriculum resources (lesson plans and other teaching materials, and Internet connectivity for student assignments)
- People resources (students, staff, and families)[1]

Once you have defined your assets you will be able assign each category a severity level. The higher the severity level the more care will need to be taken to insure its protection. Keeping these systems or networks segmented away from the lower priority networks is crucial. This would enable you to prioritize an action plan for incidents. When a new worm or exploit comes out you would want to apply patches and updates to those assets first.

"Information" resources are your highest severity assets. The networks that these systems lay upon should be completely segmented from your "People" resource assets. These systems would need to be maintained by designated administrators. Strict security practices such as patch management and OS hardening will need to become top priorities. You will need to have strong egress and ingress policies on perimeter devices. This will help prevent users from setting up rogue computers, or using unauthorized applications, such as file sharing or illegal software. The users on these networks should be required to sign acceptable usage policies, so they can be held accountable in the event that they violate the policy.

"People" resources are at the opposite end of the scale from "Information" resources. These networks are very difficult to centrally manage. Network jacks and wireless access points are nearly everywhere on a campus. There is really no way to tell who or what type of system is plugged your network from one minute to the next. Security basically falls upon the shoulders of the end user. Patching and hardening of operating system can not be easily enforced, but should be strongly recommended to all students and staff.

Now that an explanation of why segmenting the network is important

we can begin a review of the campus's current security posture.

Outbound Traffic:

There is some peer-to-peer file sharing activity which should not be allowed in a professional environment. If this is being done from University owned equipment, the University could be held liable for any illegal files on the systems, such as copyrighted material. If this activity is taking place on student owned equipment, it should be segmented away from any high severity assets. Most Universities discourage the use of file sharing but do not ban the usage, so this could be acceptable. There also appears to be a large volume of outbound web traffic which is very normal. If this is a high severity asset site, web traffic should be filtered and monitored carefully.

Inbound Traffic:

There appears to be at least one Web/FTP server on the University's network that is accessible from the Internet. This server should be segmented onto a DMZ to help keep it separate from the rest of the trusted networks. This server appears to allow anonymous FTP access. If this server is maintained by the University, it could potentially give confidential information away by allowing anyone to access the system with default credentials. Restrictive access should be placed on this server so that the anonymous account can only get files that need to be publicly available. This account should also deny the uploading of files. Misconfigured anonymous FTP servers are often used by hackers to store and trade illegal software and pornography. The University's DNS and proxy servers also appear to be getting probed from the Internet. The DNS, by people looking to find what type of server it is running. This server needs to be kept up to date with the latest vendor patches and should be configured so that it will not give any details away about itself. The proxy server is just trying to be accessed from the outside to either allow some one trusted access to the network or to allow them to hide their identity. This server appears to be configured correctly by not allowing the external user to connect and by not responding back and giving up any details about itself.

The results of this security assessment show that the overall network is in a fairly secure state. There does not appear to be any compromised or infected hosts or any known backdoors into this network. Although the several improvements mentioned should be implemented to improve the University's security posture.

Part 2: Detailed Analysis

This analysis is based on logs found at http://isc.sans.org/logs/Raw/2002.5.10

To begin my analysis I first need to find two things; the timeframe of the captured data and the basic network layout.

To determine the time frame of the dump, I grabbed the time stamps of the first and last lines of output from:

```
tcpdump -nqnr 2002.5.10 -tttt |gawk '{print $1 " " $2}'
```

Start = 06/10/2002 00:18:48.944488
Stop = 06/10/2002 23:52:36.854488

To learn the topology there are several queries that have to be run. By analyzing down to the Ethernet layer I am able to tell the basic layout of the network by using tcpdump with several specific filters.

To get the unique source MAC addresses I ran the following:

```
tcpdump -ner 2002.5.10 | awk '{print $2}' | sort -u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

Both of the addresses are associated with Cisco hardware according to a site for Vendor/Ethernet MAC Address Lookup and Search. (See http://coffer.com/mac_find/)[2]

To see how many different source addresses originate from each MAC addresses I ran the following queries with and with out the “uniq” option to get a total count and a unique count:

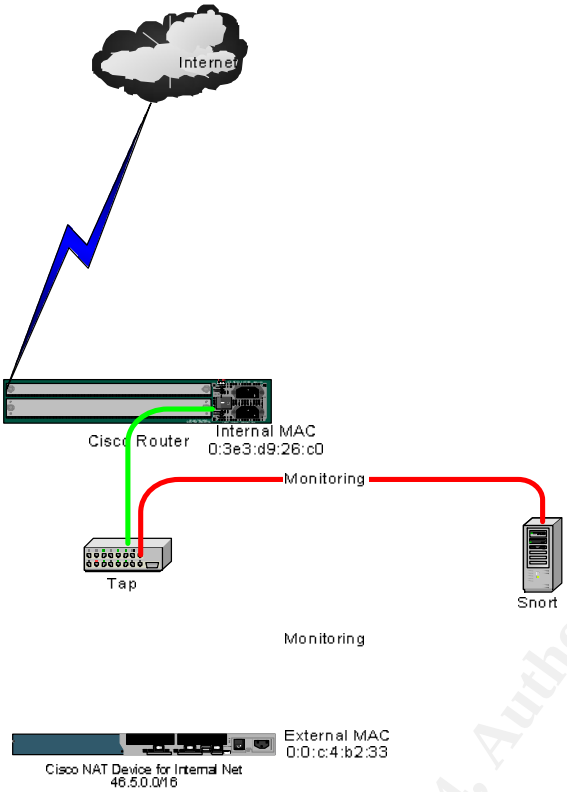
```
tcpdump -nner 2002.5.10 "ether src 0:0:c:4:b2:33" | awk '{print $6}' | awk -F\.
'{print $1 "." $2 "." $3 "." $4}' | sort -n | uniq |wc –l

tcpdump -nner 2002.5.10 "ether src 0:3:e3:d9:26:c0" | awk '{print $6}' | awk -F\.
'{print $1 "." $2 "." $3 "." $4}' | sort -n | uniq |wc -l
```

0:0:c:4:b2:33	3860 total	2 unique addresses
0:3:e3:d9:26:c0	448 total	113 unique address

The tcpdump output from the previous queries gives me a pretty good clue to the layout of the network. The external interface of the internal network is associated with the 0:0:c:4:b2:33 MAC address and 46.5.0.0/16 IP range. This device appears to be doing Network Address Translation for the 46.5.0.0/16 network. The internal interface of the external router appears to be associated with the 0:3:e3:d9:26:c0 MAC address and the Internet. The IDS

sensor is in between these two devices by means of a spanned port on a switch, a hub, or a network tap. Below is a scenario shown with a network tap.



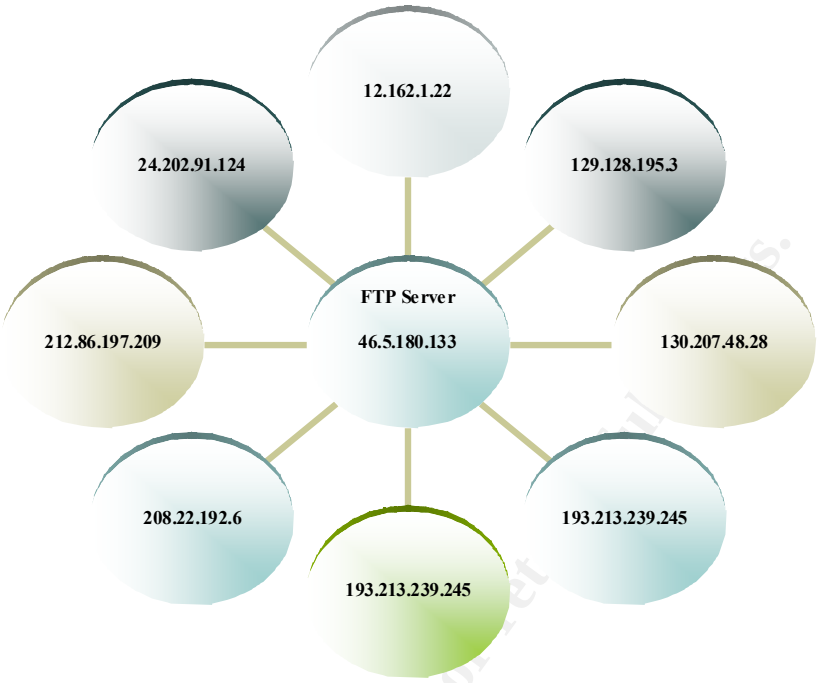
There is definitely one box that is reachable from the Internet with the public IP address of 46.4.180.133. This host is running a FTP server and an Apache Web server. We know that this box is live because it responds back with a 403 forbidden message when someone tries -to access a file they don't have permission to. To find this host I ran the following tcpdump command:

tcpdump -nnqr 2002.5.10 src net 46.5 and not host 46.5.180.250

06:00:22.784488 46.5.180.133.80 > 195.29.128.131.16453: tcp 629 (DF)
06:16:19.054488 46.5.180.133.80 > 195.29.139.104.2623: tcp 631 (DF)
06:16:26.134488 46.5.180.133.80 > 195.29.139.104.2623: tcp 630 (DF)
06:16:37.354488 46.5.180.133.80 > 195.29.139.104.2623: tcp 614 (DF)
06:16:43.364488 46.5.180.133.80 > 195.29.139.104.2623: tcp 610 (DF)
06:16:48.994488 46.5.180.133.80 > 195.29.139.104.2623: tcp 604 (DF)
06:16:54.024488 46.5.180.133.80 > 195.29.139.104.2623: tcp 599 (DF)
06:16:57.264488 46.5.180.133.80 > 195.29.139.104.2623: tcp 599 (DF)
10:11:44.304488 46.5.180.133.80 > 195.29.34.43.1057: tcp 536 (DF)
15:40:30.904488 46.5.180.133.80 > 213.240.29.118.1306: tcp 620 (DF)

This filter looks for any traffic originating from the 46.5.0.0/16 network that does not have the NAT'd address of 46.5.180.250. To see the 403 Forbidden message please see Appendix A.

Link Graph For FTP



This link graph shows a relationship of all the random hosts that attempted to log in to a FTP server (46.5.180.133) anonymously. This traffic stands out since it was in the logs but wasn't actually caught by Snort. While analyzing traffic on services that were not in my Snort results, I stumbled upon the anonymous FTP logins by running the following tcpdump query:

tcpdump –nnq –r 2002.5.10 port 21 –X

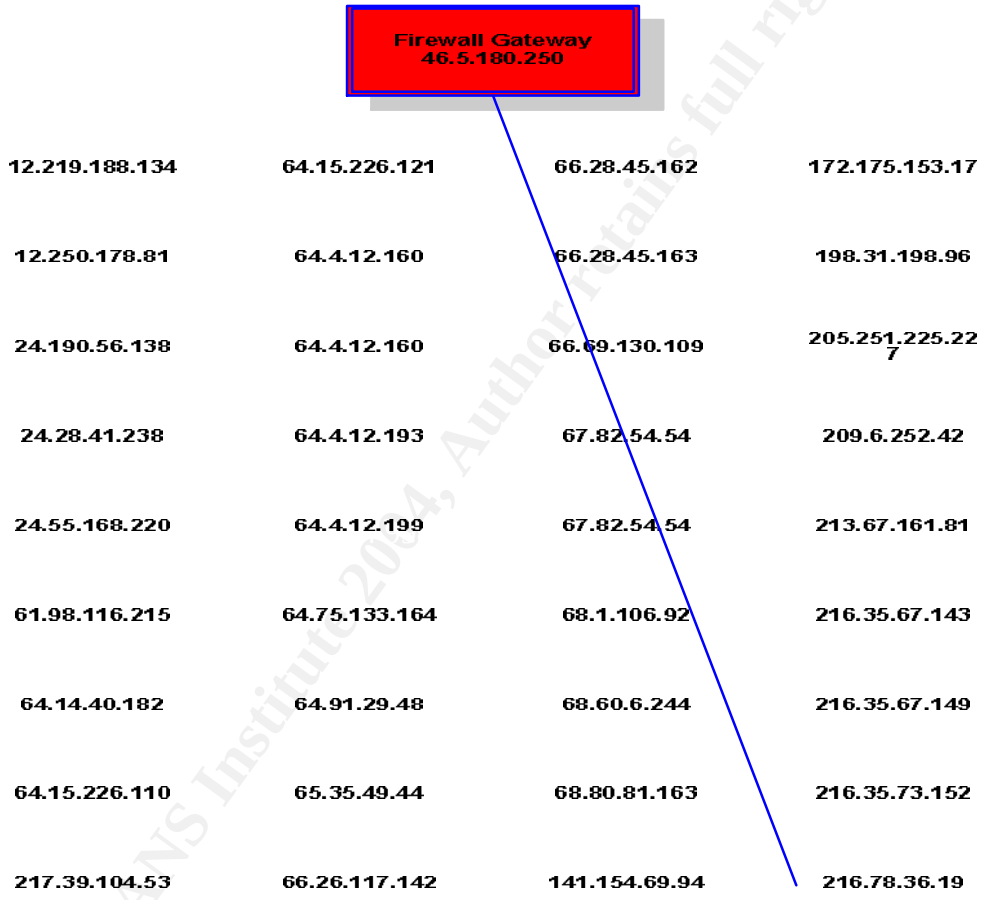
```
05:24:09.334488 193.213.239.245.49124 > 46.5.180.133.21: tcp 16 (DF)
0x0000   4500 0044 88eb 4000 2f06 3479 c1d5 eff5        E..D..@./.4y....
0x0010   2e05 b485 bfe4 0015 4bc3 b021 52cb b06a        ........K..!R..j
0x0020   8018 16d0 2840 0000 0101 080a 0d2a 44a1        ....(@.......*D.
0x0030   001d 106c 5553 4552 2061 6e6f 6e79 6d6f        ...lUSER.anonymo
0x0040   7573 0d0a                                      us..
09:58:27.584488 212.86.197.209.63730 > 46.5.180.133.21: tcp 16 (DF)
0x0000   4500 0038 34af 4000 6606 6964 d456 c5d1        E..84.@.f.id.V..
0x0010   2e05 b485 f8f2 0015 50d1 d3df 5fbf a226        ........P..._..&
0x0020   5018 4431 4d63 0000 5553 4552 2061 6e6f        P.D1Mc..USER.ano
0x0030   6e79 6d6f 7573 0d0a                            nymous..
```

Here is the Snort rule that looks for the anonymous logins:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"POLICY FTP
anonymous login attempt"; flow:to_server,established;
content:"USER"; nocase; pcre:"/^USER\s+(anonymous|ftp)/smi"; classtype:misc-
activity; sid:553; rev:7;)
```

After reviewing the rule it was pretty obvious why the alert was not triggered. The rule uses this flow option; "flow:to_server,established". This means that a connection to an internal FTP server from external client has to be established before it will run the rule against the traffic. So, since the logs don't contain the actual capture of the TCP syn, syn+ack, ack connection before the "pushed" data with the anonymous string, Snort doesn't even bother to look at the traffic.

Link Graph For P2P



This link graph shows peer to peer file sharing activity. Peer to peer file sharing, commonly know as p2p, allows users to download and upload files from other users without having to keep files stored on centrally managed systems. This graph shows one or more users behind the University's firewall being connected to 36 unique p2p hosts. This in a way bypasses firewall rules by allowing external hosts to access shared files on internal boxes. If a user were to share to the wrong directories or drives then an external user could potentially get any file on that system. The only similarity that the external users have is being connected to the same peer to peer network and needing or sharing one or more of the same files.

I was able to identify this traffic by using tcpdump and filtering out all the traffic that Snort caught and some other common traffic. Although Snort has a whole rule set devoted to p2p traffic, they don't get the chance to come into play because they also rely on the flow option which again calls for established connections.

Overview of Alerts

The following two tables show a summary of the alerts that were generated by Snort. The first chart shows the classification levels of the alerts. The second chart actual shows the alerts. The severity and priority are defined in each rule with1 being high, 2 being medium, and 3 being low.

Classifications help categorize Snort rules. The unknown severities for the http inspect and Snort decoder is because they are not based on standard rules, they are taken from decoders. The Snort decoder is what Snort uses to scrutinize TCP data, it will alert when it sees bad traffic. The HTTP Inspect decoder is best described by the following excerpt taken from the documentation that comes with Snort. Please see Appendix B for the full HTTP Inspect Readme file that also describes each individual alert that was triggered by this decoder.

HttpInspect

Daniel Roelker <droelker@sourcefire.com>

-- Overview --
HttpInspect is a generic HTTP decoder for user applications. Given a data buffer, HttpInspect will decode the buffer, find HTTP fields, and normalize the fields. HttpInspect works on both client requests and server responses.

The Distribution of Classification Method

%	No	Classification	Severity
79.34	480	http inspect	unknown
12.40	75	Attempted Information Leak	medium
7.11	43	Misc activity	low
0.99	6	Executable code was detected	high
0.17	1	snort decoder	unknown

The Distribution of Attack Methods

%	No	Attack	Priority	Severity
44.79	271	(http inspect) BARE BYTE UNICODE ENCODING {tcp}	3	unknown
25.62	155	(http inspect) BARE BYTE UNICODE ENCODING {tcp}	2	unknown
9.42	57	DNS named version attempt {udp}	2	medium
7.11	43	BACKDOOR Q access {tcp}	3	low
2.98	18	SCAN SOCKS Proxy attempt {tcp}	2	medium
2.15	13	(http inspect) OVERSIZE REQUEST-URI DIRECTORY {tcp}	2	unknown
0.99	6	(http inspect) APACHE WHITESPACE (TAB) {tcp}	3	unknown
0.99	6	(http inspect) DOUBLE DECODING ATTACK {tcp}	3	unknown
0.99	6	SHELLCODE x86 NOOP {tcp}	1	high
0.83	5	(http inspect) BARE BYTE UNICODE ENCODING {tcp}	1	unknown
0.83	5	(http inspect) DOUBLE DECODING ATTACK {tcp}	2	unknown
0.83	5	(http inspect) NON-RFC HTTP DELIMITER {tcp}	2	unknown
0.83	5	(http inspect) NON-RFC HTTP DELIMITER {tcp}	3	unknown
0.83	5	(http inspect) OVERSIZE REQUEST-URI DIRECTORY {tcp}	3	unknown
0.66	4	(http inspect) APACHE WHITESPACE (TAB) {tcp}	2	unknown
0.17	1	(snort decoder) WARNING: TCP Data Offset is less than 5! {tcp}	2	unknown

Detect1
SHELLCODE x86 NOOP

Description of Attack:

I selected to analyze the SHELLCODE x86 NOOP alerts for their serious nature, and the potential results of system compromise. The machine code for No Operations against the x86 architecture 0x90. A series of 0x90 strings together is a way for adding padding to a malicious buffer overflow exploit. A common problem when trying to execute shellcode is to know where the starting address of your shellcode should be in the stack's memory. The No Ops are instructions that delay the execution for a time period so you won't have to guess the exact offset of the starting address that would be executing your shellcode. If your shellcode begins with a large amount of No Ops you have a good chance that the offset for executing code will fall in that blank or “No Operation” memory space. After it goes through all the blank operations it will fall on your user supplied shellcode which will then be executed.

Reason this Attack was Selected:

Many different exploits and worms use the No Op feature so it is hard to make a guess at what is causing this Snort alert to be triggered. There were 5 instances of SHELLCODE x86 NOOP alerts from 205.188.212.121 to 46.4.180.250. A flag was also thrown up when noticing that the 46.4.180.250 address was involved in lots of peer to peer Gnuetlla file sharing activity. Many

worms are passed along using peer to peer networks. There was only one other instance of this alert from the IP of 128.241.244.150 directed to 46.5.180.250 on port 61653. Since there was only one packet of traffic captured from this host there wasn't really enough data to investigate deeply so I focused on the 205.188.212.121 host.

Detect was Generated By:

This detect was generated by the following Snort rule. The rule basically looks for a series of 90 90 90 in the payload of a packet. This rule can very easily produce false positives by a legitimate packet containing that same content in the payload.

$EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE x86 NOOP"; content:"|90 90 90 90 90 90 90 90 90 90 90 90 90|"; depth:128; reference:arachnids,181; classtype:shellcode-detect; sid:648; rev:7;)

This is an example of the alert that was triggered by this rule:

[**] [1:648:7] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
06/10-12:01:17.814488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x586
205.188.212.121:35594 -> 46.5.180.250:63599 TCP TTL:51 TOS:0x0 ID:26693 IpLen:20 DgmLen:1400
A* Seq: 0x6287605C Ack: 0xCD8FDEE1 Win: 0x4510 TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS181]

Tcpdump output:

In this query I am asking tcpdump to show me any instances of host 205.188.212.121 and not resolve IP addresses or port numbers. I also used the quiet/quick option to show me the least amount of protocol information. To see the payload of the packet refer to Appendix C.

tcpdump -nnq -r 2002.5.10 host 205.188.212.121

12:01:17.804488 205.188.212.121.35594 > 46.5.180.250.63599: tcp 1360
12:01:17.814488 205.188.212.121.35594 > 46.5.180.250.63599: tcp 1360
12:01:17.924488 205.188.212.121.35594 > 46.5.180.250.63599: tcp 1360
12:01:18.074488 205.188.212.121.35594 > 46.5.180.250.63599: tcp 1360
12:01:19.424488 205.188.212.121.35594 > 46.5.180.250.63599: tcp 1360

Probability the Source Address was Spoofed:

This is an ongoing TCP connection so the address is not spoofed. The address has to be real in order to complete the TCP three way handshake. As you will see below, this is a FTP session in the transfer stage. The handshake has to have been completed in order for the transfer ports to have been negotiated.

Attack Mechanism:

While doing research on the attacking host by looking for a host name I noticed that it was registered to America Online. The host's name was ftpnscp.newaol.com. From the host name it was pretty obvious that it was a legitimate FTP server. I put the name in a web browser to see if it was also running a HTTP server, it was. The page only contained a mirror of a FTP directory structure. I attempted to login to the FTP server with the anonymous account. From here it gave me the Operating System, SunOS 5.8. From this point I started thinking that the alert could possibly be a false positive that triggered when a user tried to download something. I found many reports of this signature triggering falsely via HTTP downloads, some reports on other SANS Practicals. I know this isn't the cause because I see no HTTP traffic. There is also no obvious FTP traffic either. I decided to run tcpdump on my system and then log back into ftpnscp.newaol.com. Sure enough as soon as I issued a command, my logs showed that the system veered away from the common Active FTP data port of 20 and jumped up to some high ports. I am now able to determine that the system is using passive FTP. Passive FTP works by the client creating an initial connection over TPC/21 the same as active FTP. Once the user enters in a command such as a "get" the server sends a message back to the client telling it that is entering passive mode and to connect back to the server on a specific ephemeral port to do the data transfer, not TCP/20 as in active FTP.

At this point I am able to rule out an attack and consider this a false positive caused by a user downloading something via passive FTP. Shellcode signatures can periodically trigger false positives when downloading of binary data. This traffic can be tuned out by adjusting the SHELLCODE_PORTS variable in your Snort configuration file.

Appendix D shows the tcpdump output of my passive FTP session with ftpnscp.newaol.com.

Correlations:

Since this alert is a false positive, it's hard to correlate something to this specific traffic. Here are false positives for the NOOP's signatures. It shows false positives being generated by Snort on normal web traffic.
(See http://cert.uni-stuttgart.de/archive/intrusions/2003/09/msg00114.html)[3]

Whitehats has a write up on the signature and a description for possible false positives.
(See http://www.whitehats.com/info/IDS181)[4]

Evidence of Active Targeting:

Since this traffic turned out to be a false positive, it's not truly an attack, we can reason out the active targeting.

Severity: 1

The severity is calculated in the following way: severity = (criticality +

lethality) - (system countermeasures + network countermeasures)

Criticality: 3

This appears to be a user downloading a file so chances are good that it's not an important server, it is likely just a users workstation.

Lethality: 1

At this point the user is just downloading a file. There is no way to tell what the file contains from the logs that were supplied.

System Countermeasures: 2

The system should be kept up to date with patches and the latest anti-virus signatures.

Network Countermeasures: 1

FTP traffic is allowed to egress your network. There will probably never be a time when a University will be able to totally block outbound FTP.

Detect 2: SOCKS Proxy Attempt {tcp}

Description of attack:

This attack is triggered by an external host attempting to connect to an internal proxy server. SOCKS is a proxy protocol for IP applications. A SOCKS proxy server can be used as an alternative to Squid. Proxy servers essentially open a one way door to the Internet for employees or internal users. Socks version 4 only supported TCP connections, but the new version 5 supports both TCP and UDP. A common misconfiguration is to allow anonymous or external users to connect through the server to any other hosts. Open proxies allow attackers to potential hide their identities behind the proxy server. This could allow people to tunnel through your server and launch attacks against other networks. If the proxy server is behind a firewall, this could also give an attacker access to your internal network with the identity and trust of the proxy server.

Reason this attack was selected:

I selected this alert because this traffic was only from one foreign host that attempted to connect several times to a specific IP address, specifically on SOCKS TCP port 1080. This was the only traffic captured from this host.

Detect was generated by:

This alert was triggered by the following Snort signature. This alert looks for a TCP SYN connection attempt to port 1080.

alert tcp $EXTERNAL_NET any -> $HOME_NET 1080 (msg:"SCAN SOCKS

Proxy attempt"; flags:S,12; flow:stateless;
reference:url,help.undernet.org/proxyscan/; classtype:attempted-recon; sid:615;
rev:8;)

This is a just one sample of the alerts triggered by this rule.

[**] SCAN SOCKS Proxy attempt [**]
06/10-07:47:36.994488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x4A
64.228.63.154:41554 -> 46.5.177.53:1080 TCP TTL:50 TOS:0x0 ID:40868 IpLen:20 DgmLen:60
DF
******S* Seq: 0x367901FE Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 162111066 0 NOP WS: 0

Probability the Source Address was Spoofed:

The address was not spoofed. The results from a tcpdump show that the host was waiting for the connection to come back, when it didn't, the host tried to reconnect. See Appendix E for tcpdump output for SOCKS. If the address was spoofed the host wouldn't use the standard TCP back off timer.

Attack Mechanism:

I will first give a sample of the results of a tcpdump output and then explain how the attack/scan worked.

tcpdump -nnq -r 2002.5.10 host 64.228.63.154 and src port 41554

07:47:36.994488 64.228.63.154.41554 > 46.5.177.53.1080: tcp 0 (DF)
07:47:39.984488 64.228.63.154.41554 > 46.5.177.53.1080: tcp 0 (DF)
07:47:46.004488 64.228.63.154.41554 > 46.5.177.53.1080: tcp 0 (DF)

This shows a source host at 64.228.63.154 attempting to connect to a destination of 46.5.177.53 on port 1080. We can tell that this is one TCP connection attempt by looking at the source port, and correlating it with the time of each attempt. Notice that the reconnect times are standard of that with the TCP retransmission; initial attempt, then three seconds later the second attempt, and finally the last attempt approximately six seconds later. This is commonly known as a back off timer. More details on TCP retransmission can be found in RFC 2988. (See http://www.faqs.org/rfcs/rfc2988.html) [5] Since the third attempt doesn't ever get a response, the source host figures that the destination host is not alive or is being filtered so it ends the TCP session. One second after the final third connection attempt, a new connection is started with a new source port number which attempts the standard three connections with the back off timer again. This pattern went on a total of six more times.

Correlations:

Adrian Lamo, the famous “Homeless Hacker” was known for using Internet Explorer and misconfigured proxy servers to penetrate some of the world’s largest corporations.

(See http://www.wired.com/news/culture/0,1284,50811,00.html) [6]

Snort has a detailed reference of their SCAN SOCKS Proxy attempt signature available on their signature database. In this database they reference undernet.org's page, Undernet Scans For Insecure Wingates & Proxies. (See http://help.undernet.org/proxyscan/) [7]

Evidence of Active Targeting:

This host was specifically targeted. From the logs reviewed there is no other SOCKS activity. There were only 6 separate connections attempts to 46.5.177.53 from 64.228.63.154.

Severity: 1

The severity is calculated in the following way: severity = (criticality + lethality) - (system countermeasures + network countermeasures)

Criticality: 3

A proxy server could affect a large number of valid users but usually would only affect them by not allowing access to the Internet or other external services. It wouldn't affect the public from using your web servers or sending you mail. It also wouldn't contain any sensitive data.

Lethality: 3

The damage of the scan is not all together that dangerous. The attacks that could happen if a malicious user was allowed to anonymously use your server to attack other networks could potentially be extremely high. You could be liable for any damage done to other networks originating from your server.

System Countermeasures: 3

Proper configuration is the best way to defend against misuse of your proxy severs. Not allowing relaying or external users to connect is essential to your server's security. Requiring authentication or trusted certificates would be another way to help prevent against unauthorized usage.

Network Countermeasures: 2

On a network level it would be recommend that strong filtering is placed on what services and IP addresses can reach the server.

Network Detect 3: DNS Named Version Bind Attempt

Description of Detect:

BIND (Berkeley Internet Name Domain) is an implementation of the Domain Name System (DNS) protocols. It provides named, a DNS server, and a

DNS resolver library.

This isn't so much of an attack as it is of a precursor to an attack. The attackers are in their reconnaissance phase, and are trying to find out what versions of BIND are running on particular DNS servers.

Reason this Detect was Chosen:

This detect was chosen because of the direct targeting of a service running on specific servers and for a particularly small chance of a false positive. There have been a large number of vulnerabilities exposed on Domain Name Servers running BIND.

Detect was Generated by:

This alert was triggered by the following Snort signature. The alert looks for the content "version" and "bind" to be in the payload of a packet of UDP traffic destined for port 53.

alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named version attempt"; content:"|07|version"; offset:12; nocase; content:"|04|bind"; offset:12; nocase; reference:arachnids,278; reference:nessus,10028; classtype:attempted-recon; sid:1616; rev:6;)

This is a just a sample of the 57 alerts triggered on this rule.

```
[**] DNS named version attempt [**]
06/10-05:33:34.284488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x48
203.107.136.133:1169 -> 46.5.29.29:53 UDP TTL:45 TOS:0x0 ID:45068 IpLen:20 DgmLen:58
Len: 30
12 34 00 80 00 01 00 00 00 00 00 00 07 76 65 72  .4...........ver
73 69 6F 6E 04 62 69 6E 64 00 00 10 00 03        sion.bind.....

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+

[**] DNS named version attempt [**]
06/10-10:06:51.474488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x48
203.197.102.110:4019 -> 46.5.20.184:53 UDP TTL:44 TOS:0x0 ID:55389 IpLen:20 DgmLen:58
Len: 30
12 34 00 80 00 01 00 00 00 00 00 00 07 76 65 72  .4...........ver
73 69 6F 6E 04 62 69 6E 64 00 00 10 00 03        sion.bind.....

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+
```

Below are the DNS named version attempt alerts broken down by count then by source IP.

Count	IP Address	Alert
14	203.122.47.137	DNS named version attempt {udp}
11	203.107.136.88	DNS named version attempt {udp}
9	203.197.102.110	DNS named version attempt {udp}
7	210.195.43.8	DNS named version attempt {udp}
6	203.107.136.133	DNS named version attempt {udp}
5	203.197.101.81	DNS named version attempt {udp}
3	210.195.43.28	DNS named version attempt {udp}
1	210.195.43.31	DNS named version attempt {udp}
1	203.197.101.93	DNS named version attempt {udp}

This is a small sample of the tcpdump output for UDP DNS traffic from the logs:
tcpdump -nnr 2002.5.10 udp and port 53

20:24:07.524488 203.107.136.88.3781 > 46.5.12.133.53: 4660 [b2&3=0x80] TXT CHAOS? version.bind. (30)
.....
01:35:52.244488 203.122.47.137.13605 > 46.5.37.204.53: 4660 [b2&3=0x80] TXT CHAOS? version.bind. (30)
.....
03:16:22.074488 203.107.136.88.3213 > 46.5.164.19.53: 4660 [b2&3=0x80] TXT CHAOS? version.bind. (30)
.....
19:49:20.654488 210.195.43.28.4040 > 46.5.150.2.53: 4660 [b2&3=0x80] TXT CHAOS? version.bind. (30)

Probability the Source Address was Spoofed:

The traffic is likely not spoofed; the attackers are looking for an answer to a query. In this instance it wouldn't make sense for the attacker to spoof an address, because he wouldn't get his answer back. All the queries were made from Asia, which could lead to the presumption that a group of organized attackers did this. The reason I'm assuming organized, is that none of the queries targeted the same address.

In order for spoofing to be effective in this scenario the attackers would have to be sniffing the traffic in a position directly between the DNS servers and the spoofed addresses they were using.

Attack Mechanism:

This attack or query can be achieved by issuing the UNIX command “dig (ipaddress) TXT CHAOS version.bind” or by using Nessus and issuing the command “nasl -t (ipaddress) bind.version.nasl”. There is also a plethora of scanners that can automate this scan. Once the attacker found what version of

BIND was running, he could then search for an exploit for that specific version of BIND. By default, older implementations of BIND allowed querying for the version number.

Correlations:

The ISC, Internet Software Consortium, has a very useful web page that tracks all publicly known vulnerabilities associate with BIND. (See http://www.isc.org/index.pl?/sw/bind/bind-security.php)[8]

US-CERT vulnerability note VU#196945 was released January 29, 2001. It has to do with a buffer overflow in transaction signature handing code (TSIG) in BIND version 8. (See http://www.kb.cert.org/vuls/id/196945)[9] Packet Storm has made publicly available an exploit for the BIND TSIG vulnerability called bind8x.c. (See http://www.packetstormsecurity.org/0102-exploits/bind8x.c)[10]

Evidence of Active Targeting:

The time frame of each query is rather slow which would lead me to believe these are not automated. Typically if a scan were automated you would see the attacker hitting hosts in a pattern, say climbing up a class B network from the lowest IP to the highest IP in a short period of time. The queries are suspiciously out of place. Here is an example from one host.

tcpdump -nnqr 2002.5.10 udp port 53 and host 203.107.136.133

04:23:09.434488 203.107.136.133.4302 > 46.5.139.47.53: udp 30
05:05:22.174488 203.107.136.133.3561 > 46.5.22.96.53: udp 30
05:33:34.284488 203.107.136.133.1169 > 46.5.29.29.53: udp 30
05:52:14.404488 203.107.136.133.4315 > 46.5.216.88.53: udp 30
05:53:04.084488 203.107.136.133.2252 > 46.5.246.131.53: udp 30
06:00:29.374488 203.107.136.133.3463 > 46.5.215.240.53: udp 30

This was captured over a period of 2 hours. Notice the targeted addresses are not in any order and that they don't match up with the attacker's source port. Typically, the source port should increment by one with each new connection attempt until it is reset after it reaches port 65,535. Either the host is extremely busy or someone is crafting the packets to randomly change the source ports. The 4th and 5th attempts are a lapse of 50 seconds and go through over 63,472 ports.

One possible but not likely answer could be that this is traffic from hosts infected with the li0n worm. The li0n worm was seen in early 2001, and was know for targeting vulnerable versions of BIND. The worm scanned class B networks and then checked to see what versions of BIND the servers were running. This could explain the large gaps in the source ports. There is one major problem with this theory. The infected hosts scan the class B networks for TCP/53 first to find out which servers to attack. These TCP scans do not show

up in the logs.

Severity: 1

The severity is calculated in the following way: severity = (criticality + lethality) - (system countermeasures + network countermeasures)

Criticality: 4

DNS Servers are critical to network operations but will not contain company proprietary/financial information.

Lethality: 1

This is just reconnaissance, so there is no way that that this query should effect network operations.

System Countermeasures: 3

It is unclear if the system is patched or if it is even running BIND. If the system is running BIND, a simple fix would be to configure BIND to give a null response by editing your BIND configuration file to look like this:

```
options {
        version " ";
        [ … ]
};
```

Network Countermeasures: 1

If these hosts are legitimate DNS servers this traffic will have to be allowed to pass. The queries are occurring on UDP/53, the same way a standard DNS lookup of less then 512 bytes happens.

Top Five Talking Hosts That Have Generated Alerts

%	No	IP Source	IP Destination	Attack
60.33	365	46.5.180.250	64.154.80.51	(http inspect) BARE BYTE UNICODE ENCODING {tcp}
7.93	48	46.5.180.250	64.154.80.50	(http inspect) BARE BYTE UNICODE ENCODING {tcp}
2.98	18	64.228.63.154	46.5.177.53	SCAN SOCKS Proxy attempt {tcp}
1.49	9	46.5.180.250	64.94.89.210	(http inspect) OVERSIZE REQUEST-URI DIRECTORY {tcp}
1.16	7	46.5.180.250	216.136.224.55	(http inspect) BARE BYTE UNICODE ENCODING {tcp}

This chart shows hosts that have triggered the 5 highest account of alerts between source and destination hosts. The source IP of 46.5.180.250 is the NAT address of traffic leaving the network and has triggered 4 out of the 5 highest counts. All were web based. The top 2 were to the hosts on the hitbox.com domain which are explained below. The third highest count from 46.5.180.250 went to 64.94.80.210, which at one time had the hostname of bannerserver.gator.com. (Please see http://www.brain-pro.de/outpost/Gator.txt) [12] Gator is a service that is associated with spy ware. The 4 highest from 46.5.180.250, went to 216.136.224.55(web14526.mail.yahoo.com), which is

likely just users logging into Yahoo's webmail service. The third overall was the traffic involved in detect number 2, which is associate with the SOCKS scanning.

Distribution of Events by Destination Port

%	Number of Events	Destination Port
79.34	480	80
9.42	57	53
7.11	43	515
2.98	18	1080
0.83	5	63599

HTTP and DNS are not unexpectedly the top targeted services. They are the most commonly used services and most commonly available from external networks. Port 515 is legitimately associated with being a print spooler service, although in this instance it is associated with a worm that is randomly scanning IP's for infected hosts. Port 1080 is used for SOCKS proxies. (See Detect 2) Port 63599 in this instance is being used as a data transfer port for passive FTP.

External Hosts

The first suspicious address that stands out is the broadcast source address of 255.255.255.255. This source address targeted 43 unique hosts on the 46.5.0.0 network block. This is obviously a forged TCP packet because simply enough, it can't be routed through a network. This is also the IP that is triggering the BACKDOOR Q access {tcp} alert.

Another set of suspicious addresses are 64.154.80.50 (hg1.hitbox.com) and 64.154.80.51 (ehg.hitbox.com). These servers appear to be running some sort of web/traffic analysis service called Hitbox. When browsing to the site there is a banner that says Hitbox Gateway 8.5.1 build 1. A side note, over half the web traffic in this log file is destined for these servers.

To show the unique connections to each server I ran the following tcpdump query:

```
tcpdump -nnq -r 2002.5.10 host 64.154.80.50 |awk '{print $2}' |awk -F\. '{print $1
"." $2 "." $3 "." $4 ":" $5}'|uniq |wc –l
```

285 unique connections

```
tcpdump -nnq -r 2002.5.10 host 64.154.80.51 |awk '{print $2}' |awk -F\. '{print $1
"." $2 "." $3 "." $4 ":" $5}'|uniq |wc –l
```

642 unique connections

This query attempts to show unique connections by removing duplicate entries of the source port.

There are several hosts from Asia that raise suspicion right away. These are the only hosts that are querying the DNS servers and they are all from 203.0.0.0/8 or 210.0.0.0/8 networks in Asia. They aren't seen doing anything else on the network so this appears to not be legitimate traffic.
See Appendix F for a chart of the Asian IP addresses.
See Appendix G for whois information.

Internal Hosts

According to the following tcpdump output, only 46.5.180.250 and 46.5.180.133 were the only IP's that ever had active connections to the internet.

```
tcpdump -nnq -r 2002.5.10 src net 46.5 |awk '{print $2}' |awk -F\. '{print $1 "." $2 "." $3 "." $4}' |sort |uniq

 46.5.180.133
 46.5.180.250
```

This looks for any host form the 46.5.0.0/16 network as the source address.

Since the majority of internal hosts are hidden behind the NAT address of 46.5.180.250 it's not possible to pin point any specific host as suspicious. Some of the hosts that are coming out of the network are using peer to peer file sharing applications. Other hosts could possibly have spyware installed since they were seen connecting to 64.94.80.210 (bannerserver.gator.com). The specific hosts will have be found by placing a sensor on the internal side of the network.

Correlations

DShield.org keeps an up to date database of the top ten most probed ports. (See http://www.dshield.org/topports.php) [11]
DShield.org has a write up on port 1080, which deals with scanning for SOCKS proxy servers.
(See http://dshield.com/ports/port1080.php) [13]
DShield.org has a write up on port 53, which deals with vulnerabilities in BIND.
(See http://dshield.com/ports/port53.php) [14]

"Q" is a client / server backdoor that features remote shell access. The traffic seen with the source address of 255.255.255.255 that is targeting port 515 is a known worm that looks for the "Q" Trojan. This traffic has been used for detects in several GCIA practicals.
(See www.giac.org/**practical**/ GCIA/Al_Maslowski-Yerges_GCIA.pdf)[15]
(See http://www.dshield.org/pipermail/intrusions/2002-November/005985.php) [16]

Defensive Recommendations

Proper segmentation is the key to a secure network, particularly on large networks with wide open physical access. Students, faculty, computer labs, class rooms, libraries, and business departments should all be segmented individually. The best case scenario would be to divide all the segments with firewalls and then use VPN tunnels between trusted entitles. This is the best defense for protecting systems against the spread of worms and internal attacks. Worms can't spread or people can't attack against systems that they can see.

The firewalls that protect systems that store critical data such as grades, financial and business information, and web or applications servers should have strong ingress as well as egress polices that hide them from other networks. The networks that are in use for students or computer labs could have more lenient policies that would be needed in an educational environment.

Servers that are directly facing the Internet are typically common attack targets. It is tremendously important that they stay current with the latest OS and software patches. Also, just as important, make sure to remove any services that are not specifically needed. Strong egress and ingress filtering to and from any publicly accessible servers is also a very easy way to prevent escalation in case of compromise.

One issue that is hard to get around with firewalls and segmenting is peer to peer file sharing. The clients are highly configurable and work over any port. The best solution is to use a client side IPS/firewall that will prevent the end user from even installing the software. In reality this could only be done on specific parts of the network that are trusted and under full central administration. Some University are tying to thwart the usage of p2p networks by setting bandwidth limitations and actively disconnecting users that go over their limit. Others are relying on users to sign acceptable usage policies stating that they won't use such applications and some are even trying scare users. Here is an excerpt from a memo that went to all the students at Temple University. [16]

"I'm sure you are aware that the music and motion picture industries are taking these offenses very seriously. In recent months, hundreds of students who shared files of copyrighted materials have been the targets of expensive lawsuits for copyright infringement. You may not be aware that as providers of your Internet service, Temple University is required to divulge your name and address to the authorities if a complaint is received about your illegal file-sharing activities. I want to be clear that Temple University will obey all laws and will honor all legitimate warrants, subpoenas, and court orders."

Finally, network intrusion detection sensors should be positioned on each segment on the internal network. Having sensors on the inside will help find and pinpoint compromised or infected hosts. Having a sensor outside the NAT device will cloak the internal IP addresses of your hosts. This will still let you know that something is wrong, but won't help you very much to locate the problem.

Part Three: Analysis Process

The subsequent tools were used to aid in my analysis of the Universities Network Security Infrastructure:

- Fedora Core2 2.6 Compaq PC with:
  Snort Version 2.1.3 (Build 27) with libpcap version 0.8.3
  Snortalog Version: 2.2.1
- Fedora Core1 2.4 Dell Laptop with:
  Snort Version 2.1.3 (Build 27) with libpcap version 0.7.2
  Tcpdump 3.7.2
  Ethereal 0.10.5
  Snortalog Version: 2.2.1
  OpenOffice
- Windows XP Professional
  Visio 2002
  Office 2003

www.google.com [16]

The two Fedora hosts were predominantly used to analyze all the logs. The two different Linux builds had no affect on the results. Snort and tcpdump were the main tools used, with Ethereal being used in the beginning to get a broad overview of the data. Snortalog was used to organize and display the Snort data and to help generate my charts. Snortalog is a Perl script that works by taking the output of Snort alerts and parsing the data into an organized format. (See http://jeremy.chartier.free.fr/snortalog/) [17]

My analysis began by a quick overview of the log file in ethereal. I did this to see if there were any obvious variables I could set in my Snort configuration. I was only able to come up with the net block that I could use for the HOME_NET variable.

Snort was then run with a current rule snapshot as of early July. All the rules were enabled and the HOME_NET variable was set to 46.5.0.0/16. Snort was run against the log file with the following command:

snort –c /etc/snort/snort.conf –vdek none –r 2002.5.10 –l snort-logs/

Description of commands used:
-c <rules> Use Rules File <rules>
-v Be verbose
-d Dump the Application Layer
-e Display the second layer header info
-k <mode> Checksum mode (all,noip,notcp,noudp,noicmp,none)
-r <tf> Read and proces
-l <ld> Log to directory s tcpdump file

While Snort was running on one system I started trying to put the pieces together of what exactly I would be looking into. I first determined the time frame of the logs and then moved into outlining the network structure using tcpdump. From my results I was able to draw up a network diagram, this would help to understand the outcome of the Snort alerts.

I then manually went through the logs with tcupdump doing some common queries looking for what services were running, ports were open, and publicly available servers. I then compared my results to the Snort alerts. This is when I noticed some things that Snort did not pick up on; the FTP and the Gnuetlla traffic. During this process I was able to come up with my link graphs and do most of my relationship analysis. This is also when Snortalog was used to help organize my alerts and build some of my charts. Snortalog was used with many different options and variations, please reference the Snortalog help file in Appendix H.

After this I selected three of the alerts that stood out most to me and embarked on investigating them. First I reviewed traffic from the specific alerts manually with tcpdump and then began researching them online. This is the most tedious and time consuming stage of the whole process. A misidentified alert could result in a compromised system going unnoticed, so it's the most important process, too. From here I was able to come up the information for my network detects.

Once I had the network detects finished and an overview of the network I was able to understand the firewall policy so I could come up with my defensive recommendations.

I waited until everything else was complete before approaching my executive summary. Without taking all the above steps it would impossible to come up with an accurate report as to the state of the network.

The only issue I ran into was that Snort was missing some obvious alerts. These alerts were not being triggered by Snort due to the flow option with the established variable. The established variable requires the flow be established via a stateful TCP connection before Snort will run certain rules against the data. This matter can be worked around in two different ways; either by capturing all the traffic, not just the alerts or by deep packet inspection. Capturing all the traffic is practically an impossible solution on a University size network. The way I worked through the issue was by manually going through log file and attempting to filter out the alerts that Snort had reported on and inspecting them manually. This only becomes an issue in the scenario when you have to review just a pcap file and don't have the alerts already. Chances are good that if you captured the alert in the first place you would have outputted the alerts to either an alert file or database.

The Windows workstation was only used to create the final draft, the charts, and the network diagram using Visio.

Google was used for the majority of all my web based research.

References

The following resources were referenced directly in this section or were referred to for information and not otherwise noted:

[1] Dark, Melissa and Poftak, Amy. (techlearning.com). How to Perform a Security Audit [online]. Available WWW: <URL: http://www.techlearning.com/story/showArticle.jhtml?articleID=17602668 (2004).

[2] (coffer.com) Available WWW: <URL: http://coffer.com/mac_find/

[3] Basset, Grep (cert.uni-stuttgart.de) GIAC GCIA Version 3.3 Practical Detect Available WWW: <URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/09/msg00114.html (2004).

[4] IDS181 "SHELLCODE-X86-NOPS" (Whitehats) Available WWW: <URL: *http://www.whitehats.com/info/IDS181 (2001)*

[5] Paxson, V. and Allman, M. (Network Working Group). RFC 2988 Available WWW: <URL: http://www.faqs.org/rfcs/rfc2988.html (2000).

[6] Shachtman, Noah. (Wired.com). He Hacks by Day, Squats by Night WWW: <URL http://www.wired.com/news/culture/0,1284,50811,00.html (2002).

[7] Baffy, (undernet.org) Available WWW: <URL: http://help.undernet.org/proxyscan/

[8] (Internet Systems Consortium) Available WWW: <URL: http://www.isc.org/index.pl?/sw/bind/bind-security.php (2004).

[9] (US Cert). Available WWW: <URL: http://www.kb.cert.org/vuls/id/196945

[10] Ix, Lucysoft. (Packet Storm). Available WWW: <URL: http://www.packetstormsecurity.org/0102-exploits/bind8x.c (2001)

[12] (brain-pro) Available WWW: <URL: http://www.brain-pro.de/outpost/Gator.txt

[11] (Dshield.org). Top 10 Ports. Available WWW: <URL: http://www.dshield.org/topports.php

[13] (Dshield.org). Port 1080-Proxy Server. Available WWW: <URL: http://dshield.com/ports/port1080.php

[14] (Dshield.org). Port 53-DNS Available WWW: <URL: http://dshield.com/ports/port53.php

[15] Maslowski, Yerges. (sans.org) Practical Assignment. Available WWW: <URL: www.giac.org/practical/ GCIA/Al_Maslowski-Yerges_GCIA.pdf)

[16] Szczepankiewicz, Peter. (Dshield.org). LOGS: GIAC GCIA Version 3.3 Practical Detect - Backdoor Q access. Available WWW: <URL: http://www.dshield.org/pipermail/intrusions/2002-November/005985.php

[17] O'Rourke, Timothy C. (Temple University) Temple University's Policy Regarding Peer-to-Peer File Sharing, Available WWW: <URL: http://www.temple.edu/cs/VPannouncements/filesharingpolicy.html (2003)

[18] www.google.com

[19] http://jeremy.chartier.free.fr/snortalog/

Appendix

Appendix A

403 Forbidden Message

```
06:00:22.784488 46.5.180.133.80 > 195.29.128.131.16453: tcp 629 (DF)
0x0000   4500 029d 1747 4000 3f06 01ef 2e05 b485      E....G@.?.......
0x0010   c31d 8083 0050 4045 dd8e e32f 3b1d 4f17      .....P@E.../;.O.
0x0020   5018 7b10 7d0e 0000 4854 5450 2f31 2e31      P.{.}...HTTP/1.1
0x0030   2034 3033 2046 6f72 6269 6464 656e 0d0a      .403.Forbidden..
0x0040   4461 7465 3a20 4d6f 6e2c 2031 3020 4a75      Date:.Mon,.10.Ju
0x0050   6e20 3230 3032 2030 393a 3534 3a34 3720      n.2002.09:54:47.
0x0060   474d 540d 0a53 6572 7665 723a 2041 7061      GMT..Server:.Apa
0x0070   6368 652f 312e 332e 3132 2028 556e 6978      che/1.3.12.(Unix
0x0080   2920 2028 5265 6420 4861 742f 4c69 6e75      )..(Red.Hat/Linu
0x0090   7829 206d 6f64 5f6a 6b20 6d6f 645f 7373      x).mod_jk.mod_ss
0x00a0   6c2f 322e 362e 3620 4f70 656e 5353 4c2f      l/2.6.6.OpenSSL/
0x00b0   302e 392e 3561 2050 4850 2f34 2e30 2e31      0.9.5a.PHP/4.0.1
0x00c0   706c 3220 6d6f 645f 7065 726c 2f31 2e32      pl2.mod_perl/1.2
0x00d0   3420 4672 6f6e 7450 6167 652f 342e 302e      4.FrontPage/4.0.
0x00e0   342e 330d 0a4b 6565 702d 416c 6976 653a      4.3..Keep-Alive:
0x00f0   2074 696d 656f 7574 3d31 352c 206d 6178      .timeout=15,.max
0x0100   3d31 3030 0d0a 436f 6e6e 6563 7469 6f6e      =100..Connection
0x0110   3a20 4b65 6570 2d41 6c69 7665 0d0a 5472      :.Keep-Alive..Tr
0x0120   616e 7366 6572 2d45 6e63 6f64 696e 673a      ansfer-Encoding:
0x0130   2063 6875 6e6b 6564 0d0a 436f 6e74 656e      .chunked..Conten
0x0140   742d 5479 7065 3a20 7465 7874 2f68 746d      t-Type:.text/htm
0x0150   6c3b 2063 6861 7273 6574 3d69 736f 2d38      l;.charset=iso-8
0x0160   3835 392d 310d 0a0d 0a31 3238 0d0a 3c21      859-1....128..<!
0x0170   444f 4354 5950 4520 4854 4d4c 2050 5542      DOCTYPE.HTML.PUB
0x0180   4c49 4320 222d 2f2f 4945 5446 2f2f 4454      LIC."-//IETF//DT
0x0190   4420 4854 4d4c 2032 2e30 2f2f 454e 223e      D.HTML.2.0//EN">
0x01a0   0a3c 4854 4d4c 3e3c 4845 4144 3e0a 3c54      .<HTML><HEAD>.<T
0x01b0   4954 4c45 3e34 3033 2046 6f72 6269 6464      ITLE>403.Forbidd
0x01c0   656e 3c2f 5449 544c 453e 0a3c 2f48 4541      en</TITLE>.</HEA
0x01d0   443e 3c42 4f44 593e 0a3c 4831 3e46 6f72      D><BODY>.<H1>For
0x01e0   6269 6464 656e 3c2f 4831 3e0a 596f 7520      bidden</H1>.You.
0x01f0   646f 6e27 7420 6861 7665 2070 6572 6d69      don't.have.permi
0x0200   7373 696f 6e20 746f 2061 6363 6573 7320      ssion.to.access.
0x0210   2f77 6861 7473 6e65 772f 7072 2f39 3763      /whatsnew/pr/97c
0x0220   3230 3163 6572 7470 722e 6874 6d6c 0a6f      201certpr.html.o
0x0230   6e20 7468 6973 2073 6572 7665 722e 3c50      n.this.server.<P
0x0240   3e0a 3c48 523e 0a3c 4144 4452 4553 533e      >.<HR>.<ADDRESS>
0x0250   4170 6163 6865 2f31 2e33 2e31 3220 5365      Apache/1.3.12.Se
0x0260   7276 6572 2061 7420 7777 772e 5858 5858      rver.at.www.XXXX
0x0270   2e63 6f6d 2050 6f72 7420 3830 3c2f 4144      .com.Port.80</AD
0x0280   4452 4553 533e 0a3c 2f42 4f44 593e 3c2f      DRESS>.</BODY></
0x0290   4854 4d4c 3e0a 0d0a 300d 0a0d 0a             HTML>...0....
```

Appendix B

Readme.http_inspect

HttpInspect

Daniel Roelker <droelker@sourcefire.com>

-- Overview --
HttpInspect is a generic HTTP decoder for user applications. Given a data buffer, HttpInspect will decode the buffer, find HTTP fields, and normalize the fields. HttpInspect works on both client requests and server responses.

This initial version of HttpInspect only handles stateless processing. This means that HttpInspect looks for HTTP fields on a packet by packet basis, and will be fooled if packets are not reassembled. This works fine when there is another module handling the reassembly, but there are limitations in analyzing the protocol. That's why future versions will have a stateful processing mode which will hook into various reassembly modules.

-- Configuration --
HttpInspect has a very "rich" user configuration. Users can configure individual HTTP servers with a variety of options, which should allow the user to emulate any type of web server.

It is VERY IMPORTANT to learn the configuration semantics, so you know what to expect from the normalization routines. So read this section over.

** Global Configuration **
The global configuration deals with configuration options that determine the global functioning of HttpInspect. The following example gives the generic global configuration format:

preprocessor http_inspect: global [followed by the configuration options]

You can only have a single global configuration, you'll get an error if you try otherwise.

The global configuration options are described below:

* iis_unicode_map [filename (located in the config dir)] [codemap (integer)] *
This is the global iis_unicode_map file. THIS ALWAYS NEEDS TO BE SPECIFIED IN
THE GLOBAL CONFIGURATION, otherwise you get an error. The Microsoft US
unicode codepoint map is located in the snort /etc directory as a default.
It is called unicode.map and should be used if no other is available. You can generate your own unicode maps by using the program ms_unicode_generator.c located in the HttpInspect utils directory. Remember that this configuration is for the global IIS unicode map. Individual servers can reference their own IIS unicode map.

* detect_anomalous_servers *
This global configuration option enables generic HTTP server traffic inspection on non-HTTP configured ports, and alerts if HTTP traffic is seen. DON'T turn this on if you don't have a default server configuration that encompasses all of the HTTP server ports that your users might go to. In the future we want to limit this to particular networks so it's more useful, but for right now this inspects all network traffic.

* proxy_alert *
This enables global alerting on HTTP server proxy usage. By configuring HttpInspect servers and enabling allow_proxy_use, you will only receive proxy use alerts for web users that aren't using the configured proxies or are using a rogue proxy server.

Please note that if users aren't required to configure web proxy use, then you may get a lot of proxy alerts. So, please only use this feature with traditional proxy environments. Blind firewall proxies don't count.

** Server Configuration **
This is where the fun stuff begins. There are two types of server configurations: default and [IP]. The default configuration:
- preprocessor http_inspect_server: server default [server options]

This configuration supplies the default server configuration for any server that is not individually configured. Most of your web servers will most likely end up using this default configuration. Most of the time I would suggest setting your default server to:
- preprocessor http_inspect_server: server default profile all ports { [whatever ports you want] }

In the case of individual IP's the configuration is very similar:
  - preprocessor http_inspect_server: server [IP] [server options]
Now we'll talk about the server options. Some configuration options have an argument of 'yes' or 'no'. This argument specifies whether the user wants the configuration option to generate an alert or not.

IMPORTANT:
The 'yes/no' argument does not specify whether the configuration option itself is on or off, only the alerting functionality.

* profile [all/apache/iis] *
Users can configure HttpInspect by using pre-defined HTTP server profiles. Profiles must be specified as the first server option and cannot be combined with any other options except:
  - ports
  - iis_unicode_map
  - allow_proxy_use
  - flow_depth
  - no_alerts
  - inspect_uri_only
  - oversize_dir_length
These options must be specified after the 'profile' option.

Example:

preprocessor http_inspect_server: server 1.1.1.1 profile all ports { 80 3128 }

There are three profiles available:
  - all: The "all" profile is meant to normalize the URI using most of the
                common tricks available. We alert on the more serious forms of
                evasions. This is a great profile for detecting all the types of
                attacks regardless of the HTTP server.

  - apache: The "apache" profile is used for apache web servers. This differs
                from the 'iis' profile by only excepting utf-8 standard unicode
                encoding and not excepting backslashes as legitimate slashes, like
                IIS does. Apache also excepts tabs as whitespace

  - iis: The "iis" profile mimics IIS servers. So that means we use IIS
                unicode codemaps for each server, %u encoding, bare-byte encoding,
                double decoding, backslashes, etc.

Profiles are not required by http_inspect.

* ports { [port] [port] . . . } *
This is how the user configures what ports to decode on the HTTP server.
Encrypted traffic (SSL) cannot be decoded, so adding ports 443 will only yield encoding false positives.

* iis_unicode_map [file (located in config dir(] [codemap (integer)] *
The IIS Unicode Map is generated by the program ms_unicode_generator.c. This program is located in src/preprocessors/HttpInspect/util. Executing this program generates a unicode map for the system that it was run on. So to get the specific unicode mappings for an IIS web server, you run this program on that server and use that unicode map in this configuration.

When using this option, the user needs to specify the file that contains the
IIS unicode map and also specify the unicode map to use. For US servers, this is usually 1252. But the ms_unicode_generator program tells you which codemap to use for you server, it's the ANSI codepage. You can select the correct code page by looking at the available code pages that the ms_unicode_generator outputs.

* flow_depth [integer] *
This specifies the amount of server response payload to inspect. This option significantly increases IDS performance because we are ignoring a large part of the network traffic, that we don't really have rules for anyway. Most of the HTTP server rules that we do have are for the HTTP header and a few bytes after that, so we can catch those alerts by specifying a flow_depth of about 150 -
300. Mileage may vary.

* ascii [yes/no] *
The ASCII decode option tells us whether to decode encoded ASCII chars, a.k.a %2f = /, %2e = ., etc. I suggest you don't log alerts for ASCII since it is very common to see normal ASCII encoding usage in URLs.

* utf_8 [yes/no] *
The UTF-8 decode option tells us to decode standard UTF-8 unicode sequences that are in the URI. This abides by the unicode standard and only uses % encoding. Apache uses this standard, so for any apache servers, make sure you have

this option turned on. As for alerting, you may be interested in knowing when you have an utf-8 encoded URI, but this will be prone to false positives as legitimate web clients use this type of encoding. When utf_8 is enabled, ascii decoding is also enabled to enforce correct functioning.

* u_encode [yes/no] *
This option emulates the IIS %u encoding scheme. How the %u encoding scheme works is as follows: The encoding scheme is started by a %u followed by 4 chars, like %uXXXX. The XXXX is a hex encoded value that correlates to an IIS unicode codepoint. This value can most definitely be ASCII. An ASCII char is encoded like, %u002f = /, %u002e = ., etc. If no iis_unicode_map is specified before or after this option, the default codemap is used.

You should alert on %u encodings, because I'm not aware of any legitimate clients that use this encoding. So it is most likely someone trying to be covert.

* bare_byte [yes/no] *
Bare byte encoding is an IIS trick that uses non-ASCII chars as valid values in decoding UTF-8 values. This is NOT in the HTTP standard, as all non-ASCII values have to be encoded with a %. Bare byte encoding allows the user to emulate an IIS server and interpret non-standard encodings correctly.

The alert on this decoding should be enabled, because there are no legitimate clients that encoded UTF-8 this way, since it is non-standard.

* base36 [yes/no] *
This is an option to decode base36 encoded chars. I didn't have access to a server with this option, since it appears that this is related to certain Asian versions of windows. I'm going off of info from:
http://www.yk.rim.or.jp/~shikap/patch/spp_http_decode.patch
So I hope that works for any of you with this option. Please note that if you have enabled %u encoding, this option will not work. You have to use the base36 option with the utf_8 option. Don't use the %u option, because base36 won't work. When base36 is enabled, so is ascii encoding to enforce correct behavior.

* iis_unicode [yes/no] *
The iis_unicode option turns on the unicode codepoint mapping. If there is no iis_unicode_map option specified with the server config, iis_unicode uses the default codemap. The iis_unicode option handles the mapping of non-ascii code points that the IIS server accepts and decodes normal UTF-8 request.

Users should alert on the iis_unicode option, because it is seen mainly in attacks and evasion attempts. When iis_unicode is enabled, so is ascii and utf-8 decoding to enforce correct decoding. To alert on utf-8 decoding, the user must enable also enable 'utf_8 yes'.

* double_decode [yes/no] *
The double_decode option is once again IIS specific and emulates IIS functionality. How this works is that IIS does two passes through the request
URI, doing decodes in each one. In the first pass, it seems that all types of
IIS encoding is done: UTF-8 unicode, ASCII, bare byte, and %u. In the second pass the following encodings are done: ASCII, bare byte, and %u. We leave out
UTF-8 because I think how this works is that the % encoded UTF-8 is decoded to the unicode byte in the first pass, and then UTF-8 decoded in the second stage.
Anyway, this is really complex and adds tons of different encodings for one char. When double_decode is enabled, so is ascii to enforce correct decoding.

* non_rfc_char { [byte] [0x00] . . . } *
This option let's users receive an alert if certain non-RFC chars are used in a request URI. For instance, a user may not want to see NULL bytes in the request-URI and we can give an alert on that. Please use this option with care, because you could configure it to say, alert on all '/' or something like that. It's flexible, so be careful.

* multi_slash [yes/no] *
This option normalizes multiple slashes in a row, so something like:
"foo/////////bar" get normalized to "foo/bar".

If you want an alert when multiple slashes are seen, then configure with a yes, otherwise a no.

* iis_backslash [yes/no] *
Normalize backslashes to slashes. This is again an IIS emulation. So a request-RI of "/foo\bar" gets normalized to "/foo/bar".

* directory [yes/no] *
This option normalizes directory traversals and self-referential directories. So,"/foo/this_is_not_a_real_dir/../bar" get normalized to "/foo/bar". Also, "/foo/./bar" gets normalized to "/foo/bar". If a user wants to configure an alert, then specify "yes", otherwise "no". This alert may give false positives since some web sites refer to files using directory traversals.

* apache_whitespace [yes/no] *

This option deals with non-RFC standard of tab for a space delimiter. Apache uses this, so if the emulated web server is Apache you need to enable this option. Alerts on this option may be interesting, but may also be false positive prone.

* iis_delimiter [yes/no] *
I originally started out with \n being IIS specific, but Apache takes this non-standard delimiter was well. Since this is common, we always take this as standard since the most popular web servers accept it. But you can still get an alert on this option.

* chunk_length [non-zero positive integer] *
This option is an anomaly detector for abnormally large chunk sizes. This picks up the apache chunk encoding exploits, and may also alert on HTTP tunneling that uses chunk encoding.

* no_pipeline_req *
This option turns HTTP pipeline decoding off, and is a performance enhancement if needed. By default pipeline requests are inspected for attacks, but when this option is enabled, pipeline requests are not decoded and analyzed per HTTP protocol field. It is only inspected with the generic pattern matching.

* non_strict *
This option turns on non-strict URI parsing for the broken way in which Apache servers will decode a URI. Only use this option on servers that will accept URIs like this "GET /index.html alsjdfk alsj lj aj la jsj s\n". The non_strict option assumes the URI is between the first and second space even if there is no valid HTTP identifier after the second space.

* allow_proxy_use *
By specifying this keyword, the user is allowing proxy use on this server.
This means that no alert will be generated if the proxy_alert global keyword has been used. If the proxy_alert keyword is not enabled, then this option does nothing. The allow_proxy_use keyword is just a way to suppress unauthorized proxy use for an authorized server.

* no_alerts *
This option turns off all alerts that are generated by the HttpInspect preprocessor module. This has no effect on http rules in the ruleset.
No argument is specified.

* oversize_dir_length [non-zero positive integer] *
This option takes a non-zero positive integer as an argument. The argument specifies the max char directory length for URL directory. If a URL directory is larger than this argument size, an alert is generated.
A good argument value is 300 chars. This should limit the alerts to IDS evasion type attacks, like whisker -I 4.

* inspect_uri_only *
This is a performance optimization. When enabled, only the URI portion of HTTP requests will be inspected for attacks. As this field usually contains 90-95% of the web attacks, you'll catch most of the attacks. So if you need extra performance, then enable this optimization. It's important to note that if this option is used without any uricontent rules, then no inspection will take place. This is obvious since the uri is only inspected with uricontent rules, and if there are none available then there is nothing to inspect.

For example, if we have the following rule set:

alert tcp any any -> any 80 (msg:"content"; content: "foo";)

and the we inspect the following URI:

GET /foo.htm HTTP/1.0\r\n\r\n

No alert will be generated when 'inspect_uri_only' is enabled. The 'inspect_uri_only' configuration turns off all forms of detection except uricontent inspection.

-- Profile Breakout --
There are three profiles that users can select. Only the configuration that are listed under the profiles are turned on. If there is no mention of alert on or off, then that means there is no alert associated with the configuration.

* Apache *

flow_depth 300
chunk encoding (alert on chunks larger than 500000 bytes)
ascii decoding is on (alert off)
looking for NULL bytes in URL (alert on)
multiple slash (alert off)
directory normalization (alert off)
apache whitespace (alert on)
utf_8 encoding (alert off)

non_strict URL parsing

* IIS *

flow_depth 300
iis_unicode_map is set to the codepoint map in the global configuration
ascii decoding (alert off)
multiple slash (alert off)
directory normalization (alert off)
double decoding (alert on)
%u decoding (alert on)
bare byte decoding (alert on)
iis unicode codepoints (alert on)
iis backslash (alert off)
iis delimiter (alert on)

* All *

flow_depth 300
chunk encoding (alert on chunks larger than 500000 bytes)
iis_unicode_map is set to the codepoint map in the global configuration
ascii decoding is on (alert off)
looking for NULL bytes in URL (alert on)
multiple slash (alert off)
directory normalization (alert off)
apache whitespace (alert on)
double decoding (alert on)
%u decoding (alert on)
bare byte decoding (alert on)
iis unicode codepoints (alert on)
iis backslash (alert off)
iis delimiter (alert on)

-- Writing uricontent rules --
The uricontent parameter in the snort rule language searches the NORMALIZED request URI field. This means that if you are writing rules that include things that are normalized, such as %2f or directory traversals, these rules will not alert. The reason is that the things you are looking for are normalized out of the URI buffer. For example, the URI:

/scripts/..%c0%af../winnt/system32/cmd.exe?/c+ver

will get normalized into:

/winnt/system32/cmd.exe?/c+ver

Another example,

/cgi-bin/aaaaaaaaaaaaaaaaaaaaaaaa/..%252fp%68f?

into:

/cgi-bin/phf?

So when you are writing a uricontent rule, you should write the content that you want to find in the context that the URI will be normalized. Don't include directory traversals (if you normalize directories) and don't look for encode characters. You can accomplish this type of detection by using the 'content' rule parameter, since this rule inspects the unnormalized buffer.

-- Conclusion --
So you got to the end? Good for you. I'm sure you know more about HTTP encodings and evasions then you ever wanted to. My suggestions are to stick with the "profile" options, since they are much easier to read and have been researched.

If you feel like giving us profiles for other web servers, please do.
We'll incorporate them into the default server profiles for HttpInspect.

Appendix C

ShellCode Payload

Tcpdump –nnq –r 2002.10.5 host 205.188.212.121 –X –c 1

```
12:01:17.804488 205.188.212.121.35594 > 46.5.180.250.63599: tcp 1360
0x0000  4500 0578 6844 0000 3306 9b0c cdbc d479  E..xhD..3......y
0x0010  2e05 b4fa 8b0a f86f 6287 5b0c cd8f dee1  ........ob.[.....
0x0020  5018 4510 43cd 0000 558b ec8b 4510 568b  P.E.C...U...E.V.
0x0030  750c 8b4d 0857 8d7c 30ff 8bc7 3bc6 7212  u..M.W.|0...;.r.
0x0040  8ad1 80e2 0780 c230 8810 c1f9 0348 3bc6  .......0.....H;.
0x0050  73ee 3bf7 8bc6 730d 8038 3075 08c6 0020  s.;...s..80u....
0x0060  403b c772 f35f 5e5d c390 9090 9090 9090  @;.r._^].........
0x0070  9090 9090 9090 9090 558b ec8b 4508 568b  ........U...E.V.
0x0080  7510 578b 7d0c 5657 50e8 9aff ffff 83c4  u.W.}.VWP.......
0x0090  0cc6 043e 20c6 443e 0100 5f5e 5dc3 9090  ...>..D>.._^]...
0x00a0  9090 9090 9090 9090 558b ec57 8b7d 0c33  ........U..W.}.3
0x00b0  c033 d285 ff7e 2556 8b75 088a 0c32 80f9  .3...~%V.u...2..
0x00c0  3072 1380 f937 770e 81e1 ff00 0000 c1e0  0r...7w.........
0x00d0  0383 e930 0bc1 423b d77c e05e 5f5d c390  ...0..B;.|.^_]..
0x00e0  9090 9090 9090 9090 558b ec83 ec30 568d  ........U....0V.
0x00f0  4df4 e861 2f00 008b 4d0c 8b55 088d 45f4  M..a/...M..U..E.
0x0100  5051 52e8 7000 0000 8b45 1083 c40c 8d4d  PQR.p....E.....M
0x0110  f450 e831 3400 008b 4d14 85c9 7409 8d55  .P.14...M...t..U
0x0120  f452 e8e1 3100 008b 4df4 8d45 d050 51e8  .R..1...M..E.PQ.
0x0130  746e 0100 83c4 0883 f8ff 740f 8d4d f4e8  tn........t..M..
0x0140  3430 0000 33c0 5e8b e55d c38b 55f4 6878  40..3.^..]..U.hx
0x0150  1142 0052 e81f 5000 0083 c408 8d4d f48b  .B.R..P......M..
0x0160  f0e8 1230 0000 8bc6 5e8b e55d c390 9090  ...0....^..]....
0x0170  9090 9090 9090 9090 558b ec81 ec14 0400  ........U.......
0x0180  008b 450c 5356 5780 3800 741d 8b5d 086a  ..E.SVW.8.t..].j
0x0190  5c50 538d 85ec fbff ff68 a412 4200 50e8  \PS......h..B.P.
0x01a0  a445 0000 83c4 14eb 288b 5d08 83c9 ff8b  .E......(.].....
0x01b0  fb33 c0f2 aef7 d12b f98d 95ec fbff ff8b  .3.....+........
0x01c0  c18b f78b fac1 e902 f3a5 8bc8 83e1 03f3  ................
0x01d0  a48b fb83 c9ff 33c0 6a5c f2ae f7d1 498d  ......3.j\....I.
0x01e0  840d ecfb ffff 50e8 5c2b 0000 8bf0 83c4  ......P.\+......
0x01f0  0885 f674 4f8d 4dec c606 00e8 880e 0000  ...tO.M.........
0x0200  8d8d ecfb ffff 518d 4dec e8c9 0e00 008d  ......Q.M.......
0x0210  4dec e891 1100 0085 c075 0c8d 4dec e865  M........u..M..e
0x0220  1100 0085 c074 398d 4dec c606 5ce8 760e  .....t9.M...\.v.
0x0230  0000 466a 5c56 e80d 2b00 008b f083 c408  ..Fj\V..+.......
0x0240  85f6 75b1 8b4d 1085 c974 0c8d 95ec fbff  ..u..M...t......
0x0250  ff52 e821 3100 0033 c05f 5e5b 8be5 5dc3  .R.!1..3._^[..].
0x0260  8d4d ece8 400e 0000 5f5e 33c0 5b8b e55d  .M..@..._^3.[..]
0x0270  c390 9090 9090 9090 558b ec83 ec50 5356  ........U....PSV
0x0280  578d 4db0 33ff e8fd 0d00 008b 7508 8d4d  W.M.3.......u..M
0x0290  d056 e851 2f00 0056 8d4d b0e8 380e 0000  .V.Q/..V.M..8...
0x02a0  8d4d b0e8 0011 0000 85c0 756c 8d4d b0e8  .M........ul.M..
0x02b0  b416 0000 8b45 b48d 4ddc 50e8 282f 0000  .....E..M.P.(/..
0x02c0  8b4d dc8b 41f8 8d4d f48d 7801 e887 2d00  .M..A..M..x...-.
0x02d0  0080 3e00 741a 8b55 dc56 6a5c 528d 45f4  ..>.t..U.Vj\R.E.
0x02e0  6824 1142 0050 e8ed 3400 0083 c414 eb0c  h$.B.P..4.......
0x02f0  8d4d dc51 8d4d f4e8 0c30 0000 8b55 f48d  .M.Q.M...0...U..
0x0300  4db0 52e8 d00d 0000 8d4d f4e8 682e 0000  M.R......M..h...
0x0310  8d4d dce8 602e 0000 8d4d e8e8 382d 0000  .M..`....M..8-..
0x0320  6804 0100 0068 0501 0000 8d4d e8e8 5632  h....h.....M..V2
0x0330  0000 508b 450c 508d 4db0 e859 1100 006a  ..P.E.P.M..Y...j
0x0340  ff8d 4de8 8bf0 e8bd 3200 0083 fe02 0f84  ..M.....2.......
0x0350  0f01 0000 8b5d 108b c78b 7d14 25ff ff00  .....]....}.%...
0x0360  0089 4508 eb03 8b45 088b 4de8 8b51 f88d  ..E....E..M..Q..
0x0370  4de8 2bd0 8d45 f452 50e8 4a33 0000 83ee  M.+..E.RP.J3....
0x0380  0074 3f4e 0f85 a100 0000 8b4d f453 688c  .t?N.......M.Sh.
0x0390  4d42 0051 ffd7 8bf0 83c4 0c85 f60f 8ce1  MB.Q............
0x03a0  0000 008b 55f4 5753 68ac 1242 0052 e8c5  ....U.WSh..B.R..
0x03b0  feff ff8b f083 c410 85f6 0f8c c400 0000  ................
0x03c0  eb69 6a5c 8d4d f4e8 ac33 0000 8bf0 83fe  .ij\.M...3......
0x03d0  ff75 138b 45f4 5350 688c 4d42 00ff d783  .u..E.SPh.MB....
```

0x03e0	c40c 8bf0 eb41 8b4d f48d 45c4 8b51 f88d	.....A.M..E..Q..
0x03f0	4df4 2bd6 4a52 50e8 cc32 0000 8b00 5350	M.+.JRP..2....SP
0x0400	8d4d dc56 518d 4df4 e81b 3300 008b 1052	.M.VQ.M...3....R
0x0410	ffd7 83c4 0c8d 4ddc 8bf0 e859 2d00 008d	......M....Y-...
0x0420	4dc4 e851 2d00 0085 f67c 5968 0401 0000	M..Q-....|Yh....
0x0430	6805 0100 008d 4de8 e84b 3100 0050 8d4d	h.....M..K1..P.M
0x0440	b0e8 2212 0000 6aff 8d4d e88b f0e8 b631	.."...j..M.....1
0x0450	0000 8d4d f4e8 1e2d 0000 83fe 020f 8503	...M...-........
0x0460	ffff ff8d 4de8 e80d 2d00 008d 4dd0 e805	....M...-...M...
0x0470	2d00 008d 4db0 e82d 0c00 0033 c05f 5e5b	-...M..-...3._^[
0x0480	8be5 5dc3 8d4d f4e8 ec2c 0000 8d4d e8e8	..]..M...,...M..
0x0490	e42c 0000 8d4d d0e8 dc2c 0000 8d4d b0e8	.,...M...,...M..
0x04a0	040c 0000 8bc6 5f5e 5b8b e55d c390 **9090**	......_^[..]....
0x04b0	**9090 9090 9090 9090** 558b ec56 578b 7d10	........U..VW.}.
0x04c0	85ff 750b b803 4000 805f 5e5d c20c 008b	..u...@.._^]....
0x04d0	450c 8b15 e0e2 4100 8b08 3bd1 753e 8b15	E.....A...;.u>..
0x04e0	e4e2 4100 8b70 043b d675 318b 15e8 e241	..A..p.;.u1....A
0x04f0	008b 7008 3bd6 7524 8b15 ece2 4100 8b70	..p.;.u$....A..p
0x0500	0c3b d675 178b 7508 8b4e fc8d 46fc 50ff	.;.u..u..N..F.P.
0x0510	5104 8937 33c0 5f5e 5dc2 0c00 390d 80e7	Q..73._^]...9...
0x0520	4100 753e 8b15 84e7 4100 8b48 043b d175	A.u>....A..H.;.u
0x0530	318b 0d88 e741 008b 5008 3bca 7524 8b15	1....A..P.;.u$..
0x0540	8ce7 4100 8b48 0c3b d175 178b 4508 8b48	..A..H.;.u..E..H
0x0550	fc8d 70fc 56ff 5104 8937 33c0 5f5e 5dc2	..p.V.Q..73._^].
0x0560	0c00 c707 0000 0000 5fb8 0240 0080 5e5d	........_..@..^]
0x0570	c20c 00**90 9090 9090**	.....

Appendix D

My FTP Session

Tcpdump –nn host 205.188.212.121

17:30:06.646208 192.168.1.101.33686 > 205.188.212.121.21: tcp 0
17:30:06.658650 205.188.212.121.21 > 192.168.1.101.33686: tcp 0
17:30:06.658671 192.168.1.101.33686 > 205.188.212.121.21: tcp 0
17:30:06.684396 205.188.212.121.21 > 192.168.1.101.33686: tcp 8
17:30:06.684425 192.168.1.101.33686 > 205.188.212.121.21: tcp 0
17:30:06.697014 205.188.212.121.21 > 192.168.1.101.33686: tcp 54
17:30:06.697038 192.168.1.101.33686 > 205.188.212.121.21: tcp 0
17:30:11.316009 192.168.1.101.33686 > 205.188.212.121.21: tcp 16
17:30:11.331704 205.188.212.121.21 > 192.168.1.101.33686: tcp 0
17:30:11.333129 205.188.212.121.21 > 192.168.1.101.33686: tcp 68
17:30:11.333141 192.168.1.101.33686 > 205.188.212.121.21: tcp 0
17:30:14.253041 192.168.1.101.33686 > 205.188.212.121.21: tcp 20
17:30:14.270316 205.188.212.121.21 > 192.168.1.101.33686: tcp 0
17:30:14.271261 205.188.212.121.21 > 192.168.1.101.33686: tcp 48
17:30:14.283930 192.168.1.101.33686 > 205.188.212.121.21: tcp 0
17:30:14.293887 192.168.1.101.33686 > 205.188.212.121.21: tcp 6
17:30:14.340744 205.188.212.121.21 > 192.168.1.101.33686: tcp 34
17:30:14.373874 192.168.1.101.33686 > 205.188.212.121.21: tcp 0
17:30:16.060820 192.168.1.101.33686 > 205.188.212.121.21: tcp 6
17:30:16.075099 205.188.212.121.21 > 192.168.1.101.33686: tcp 53
17:30:16.083879 192.168.1.101.33686 > 205.188.212.121.21: tcp 0
17:30:16.083954 192.168.1.101.33687 > 205.188.212.121.38555: tcp 0
17:30:16.102161 205.188.212.121.38555 > 192.168.1.101.33687: tcp 0
17:30:16.102193 192.168.1.101.33687 > 205.188.212.121.38555: tcp 0
17:30:16.102232 192.168.1.101.33686 > 205.188.212.121.21: tcp 6
17:30:16.125387 205.188.212.121.21 > 192.168.1.101.33686: tcp 53
17:30:16.140724 205.188.212.121.38555 > 192.168.1.101.33687: tcp 609
17:30:16.140773 192.168.1.101.33687 > 205.188.212.121.38555: tcp 0
17:30:16.141118 205.188.212.121.38555 > 192.168.1.101.33687: tcp 0
17:30:16.143876 192.168.1.101.33687 > 205.188.212.121.38555: tcp 0
17:30:16.158331 205.188.212.121.38555 > 192.168.1.101.33687: tcp 0
17:30:16.163867 192.168.1.101.33686 > 205.188.212.121.21: tcp 0
17:30:16.179383 205.188.212.121.21 > 192.168.1.101.33686: tcp 30

17:30:16.179430 192.168.1.101.33686 > 205.188.212.121.21: tcp 0

Appendix E

SOCKS Tcpdump Output

07:47:36.994488 64.228.63.154.41554 > 46.5.177.53.1080: tcp 0 (DF)
07:47:39.984488 64.228.63.154.41554 > 46.5.177.53.1080: tcp 0 (DF)
07:47:46.004488 64.228.63.154.41554 > 46.5.177.53.1080: tcp 0 (DF)
07:47:47.234488 64.228.63.154.44086 > 46.5.177.53.1080: tcp 0 (DF)
07:47:50.254488 64.228.63.154.44086 > 46.5.177.53.1080: tcp 0 (DF)
07:47:56.004488 64.228.63.154.44086 > 46.5.177.53.1080: tcp 0 (DF)
07:47:56.994488 64.228.63.154.46539 > 46.5.177.53.1080: tcp 0 (DF)
07:48:00.104488 64.228.63.154.46539 > 46.5.177.53.1080: tcp 0 (DF)
07:48:06.134488 64.228.63.154.46539 > 46.5.177.53.1080: tcp 0 (DF)
07:48:07.094488 64.228.63.154.49139 > 46.5.177.53.1080: tcp 0 (DF)
07:48:09.994488 64.228.63.154.49139 > 46.5.177.53.1080: tcp 0 (DF)
07:48:16.174488 64.228.63.154.49139 > 46.5.177.53.1080: tcp 0 (DF)
07:48:17.604488 64.228.63.154.51449 > 46.5.177.53.1080: tcp 0 (DF)
07:48:20.004488 64.228.63.154.51449 > 46.5.177.53.1080: tcp 0 (DF)
07:48:26.014488 64.228.63.154.51449 > 46.5.177.53.1080: tcp 0 (DF)
07:48:26.994488 64.228.63.154.53744 > 46.5.177.53.1080: tcp 0 (DF)
07:48:30.004488 64.228.63.154.53744 > 46.5.177.53.1080: tcp 0 (DF)
07:48:35.994488 64.228.63.154.53744 > 46.5.177.53.1080: tcp 0 (DF)

Appendix F

Suspicious Asian External Addresses

Count	IP	Country
14	203.122.47.137	Asia
11	203.107.136.88	Asia
9	203.197.102.110	Asia
7	210.195.43.8	Asia
6	203.107.136.133	Asia
5	203.197.101.81	Asia
3	210.195.43.28	Asia
1	210.195.43.31	Asia
1	203.197.101.93	Asia

Appendix G

Whois Information

```
whois 64.154.80.51
[Querying whois.arin.net]
[whois.arin.net]

OrgName:    Level 3 Communications, Inc.
OrgID:      LVLT
Address:    1025 Eldorado Blvd.
City:       Broomfield
StateProv:  CO
PostalCode: 80021
Country:    US

NetRange:   64.152.0.0 - 64.159.255.255
CIDR:       64.152.0.0/13
NetName:    LC-ORG-ARIN
NetHandle:  NET-64-152-0-0-1
Parent:     NET-64-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.LEVEL3.NET
NameServer: NS2.LEVEL3.NET
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-
PORTABLE
RegDate:    2000-06-08
Updated:    2001-05-30

TechHandle: LC-ORG-ARIN
TechName:   level Communications
TechPhone:  +1-877-453-8353
TechEmail:  ipaddressing@level3.com

OrgAbuseHandle: APL8-ARIN
OrgAbuseName:   Abuse POC LVLT
OrgAbusePhone:  +1-877-453-8353
OrgAbuseEmail:  abuse@level3.com

OrgTechHandle: TPL1-ARIN
OrgTechName:   Tech POC LVLT
OrgTechPhone:  +1-877-453-8353
OrgTechEmail:  ipaddressing@level3.com

OrgTechHandle: ARINC4-ARIN
OrgTechName:   ARIN Contact
OrgTechPhone:  +1-800-436-8489
OrgTechEmail:  arin-contact@genuity.com

# ARIN WHOIS database, last updated 2004-08-25 19:10
# Enter ? for additional hints on searching ARIN's WHOIS
database.
```

```
whois 203.122.47.137
[Querying whois.apnic.net]
[whois.apnic.net]
% [whois.apnic.net node-1]
% Whois data copyright terms
http://www.apnic.net/db/dbcopyright.html

inetnum:      203.122.47.0 - 203.122.47.255
netname:      SHARED-DSL-OKH-II
country:      IN
descr:        Pool of IP's dynamically assigned to DSL
routers of Okhla
descr:        email : j.grewal@in.spectranet.com
admin-c:      PS272-AP
tech-c:       JG131-AP
status:       ASSIGNED NON-PORTABLE
changed:      harpreet.singh@in.spectranet.com
20040323
notify:       apnic.admin@in.spectranet.com
mnt-by:       MAINT-IN-SPECTRA-NET-LTD
source:       APNIC

person:       Pawan Pratap Singh
address:      42-Okhla Industrial Estate
address:      Phase - III
address:      New Delhi
country:      IN
phone:        +91-11-6200872
fax-no:       +91-11-6200805
e-mail:       pawan.singh@in.spectranet.com
nic-hdl:      PS272-AP
mnt-by:       MAINT-IN-SPECTRANET
changed:      sanjeev.sharma@in.spectranet.com
20020703
source:       APNIC

person:       J S Grewal
address:      42-Okhla Industrial Estate
address:      Phase - III
address:      New Delhi
country:      IN
phone:        +91-11-6200876
fax-no:       +91-11-6200805
e-mail:       j.grewal@in.spectranet.com
nic-hdl:      JG131-AP
mnt-by:       MAINT-IN-SPECTRANET
changed:      sanjeev.sharma@in.spectranet.com
20020703
source:       APNIC
```

Appendix G Cont.

```
whois 210.195.43.8
[Querying whois.apnic.net]
[whois.apnic.net]
% [whois.apnic.net node-1]
% Whois data copyright terms
http://www.apnic.net/db/dbcopyright.html

inetnum:     210.195.0.0 - 210.195.63.255
netname:     INFRA-TMNET
descr:       TMNET
country:     MY
admin-c:     TA35-AP
tech-c:      TA35-AP
mnt-by:      TM-NET-AP
changed:     anieayop@tm.net.my 20040409
status:      ASSIGNED NON-PORTABLE
source:      APNIC

role:        TMNET IP Administrators
address:     Level 25 (South), Menara Telekom,
address:     Jalan Pantai Baru,
address:     50672 Kuala Lumpur.
country:     MY
phone:       +603-22403034
fax-no:      +603-22403034
e-mail:      e_melia@tm.net.my
e-mail:      anieayop@tm.net.my
e-mail:      aizan98@tm.net.my
e-mail:      fuwaizah@tm.net.my
trouble:     abuse@tm.net.my
trouble:     streamyx@tm.net.my
trouble:     noc@tm.net.my
trouble:     tmcops@tm.net.my
trouble:     dnsteam@tm.net.my
admin-c:     EU3-AP
admin-c:     NA31-AP
tech-c:      EU3-AP
tech-c:      NA31-AP
tech-c:      SS456-AP
tech-c:      SM135-AP
nic-hdl:     TA35-AP
mnt-by:      TM-NET-AP
changed:     anieayop@tm.net.my 20040415
source:      APNIC
```

```
whois 203.107.136.88
[Querying whois.apnic.net]
[whois.apnic.net]
% [whois.apnic.net node-2]
% Whois data copyright terms
http://www.apnic.net/db/dbcopyright.html

inetnum:     203.107.128.0 - 203.107.255.255
netname:     COMNET-TH
descr:       KSC Commercial Internet Co. Ltd.
descr:       2/4 Samaggi Insurance Tower 10th Fl.,
descr:       Viphavadee-Rangsit RD
descr:       Thungsonghong, Laksi
descr:       Bangkok 10210
country:     TH
admin-c:     JT183-AP
tech-c:      TOC1-AP
remarks:     service provider
remarks:     Delegate four /19 to /17
mnt-by:      APNIC-HM
mnt-lower:   KSC-ADMIN
changed:     hostmaster@apnic.net 20000301
changed:     hostmaster@apnic.net 20011016
changed:     hm-changed@apnic.net 20020830
status:      ALLOCATED PORTABLE
source:      APNIC

person:      Joost Th.A Doevelaar
nic-hdl:     JT183-AP
e-mail:      jdoevelaar@ksc.net
address:     KSC Commercial Internet Co.,Ltd.
address:     2/4 Samaggi Insurance Tower 10th Fl., Viphavadee-
Rangsit Rd.,
address:     Thungsonghong, Laksi
address:     Bangkok 10210
phone:       +66-2-9797777
fax-no:      +66-2-5885665
country:     TH
changed:     netadmin@ns.ksc.co.th 20030115
mnt-by:      KSC-ADMIN
source:      APNIC

person:      Technical Operation Center
address:     KSC Commercial Internet Co.,Ltd.
address:     Operation Department
address:     2/4 Samaggi Insurance Tower 10th Fl., Viphavadee-
Rangsit Rd.,
address:     Thungsonghong, Laksi
address:     Bangkok 10210
```

Appendix H

Snortalog Help

```
snortalog --help
Unknown option: help
Usage: cat <alerts file> or <snort.rules> | /usr/bin/snortalog <options> <reports> <filters>
 Options:
-x                    Mode GUI
-r                    Resolve IP adresses
-c                    Resolve domains
-h <file.html>          Specify a HTML file
-p <file.pdf>           Specify a PDF file
-u <directory>           Specify an output directory
-g <gif|png|jpg>         Graph output format
-i                   Inverse the result
-d                    Mode debug
-n <integer>            Specify a number of line in the result
-file <log file>        Specify an input alert log file
-rulesfile <file>       Specify name and directory to search rules file
-hwfile <file>          Specify name and directory to search hardware file
-domainsfile <file>       Specify name and directory to search domains file
-genref <rules file>      Generate the reference rules file
-help                  View this help
 Reports:
-src                  Top IPs sources
-dst                  Top IPs destination
-src_attack             Top IPs sources grouped by attack
-dst_attack             Top IPs destination grouped by attack
-src_dst_attack          Top alert grouped by IPs sources, Ips destination and attack
-attack                Top attack
-class                 Top classification
-severity              Top severity
-daily_event             Top number of attack grouped by day
-hour                  Top number of attack grouped by hour
-hour_attack             Top specific attack grouped by hour
-dport                 Top destination port
-proto                 Top protocols
-dport_attack            Top destination port grouped by attack
-nids                  Top NIDS host
-stateful              Top stateful problems
-interfaces             Top interfaces events
-domain_src              Top of domain source
-portscan               Top of portscan alert
-actions               Top of firewall action (DROP, REJECT, ACCEPT, etc ...)
-rules                 Top of rule (only Fw-1)
-reasons               Top of reason (only Fw-1)
-src_dport             Top IPs sources grouped by destination port
-dst_dport             Top IPs destination grouped by destination port
-typelog               Number of occurrences by type of log
-hwlog                 Number of occurrences by hardware related message log<
-report                All reports
 Filters:
-fsrc                 Sources filter
-fdst                 Destination filter
-fproto                Protocol filter
-fdport                Destination port filter
-fmonth                 Month filter
-fday                  Day filter
-fhour                 Hour filter
-fether                Interface filter
-fseverity              Severity filter
-faction               Firewall action filter
-frule                 Firewall rule filter
-ftype                 Type of logs

Version: 2.2.1
Jeremy CHARTIER, <jeremy.chartier@free.fr>
Date: 2004/05/03 17:19:00
```