Global Information Assurance Certification Paper

GCIA Practical Assignment version 3.3

Tyler Hudak

Table of Contents

Paper Conventions

This following conventions are used within this paper.

Arial 12 font is used for normal text.

The Times New Roman 10 or `Courier New 10` fonts are used for packet information.

Open Proxy Server Scans

Executive Summary

Every so often Intrusion Detection Systems or firewalls will detect scans for TCP ports 80, 1080, 3128 or 8080 across a network, such as the slow scan shown below.

```
2002/12/26-07:26:18.885979 scanner.5.1668 > home.68.1080: S 1618127438:1618127438(0) win 1024
2002/12/26-07:26:20.778587 scanner.5.45514 > home.68.3128: S 1642554383:1642554383(0) win 1024
2002/12/26-07:26:22.642234 scanner.5.61494 > home.68.8080: S 449166470:449166470(0) win 1024
2002/12/26-07:26:24.571208 scanner.5.30426 > home.68.80: S 278315363:278315363(0) win 1024
2002/12/26-14:41:59.535045 scanner.5.39184 > home.34.1080: S 83523683:83523683(0) win 1024
2002/12/26-14:42:01.393228 scanner.5.19351 > home.34.3128: S 1163112705:1163112705(0) win 1024
2002/12/26-14:42:03.265129 scanner.5.64907 > home.34.8080: S 237470119:237470119(0) win 1024
2002/12/26-14:42:05.143377 scanner.5.18712 > home.34.80: S 598834575:598834575(0) win 1024
2002/12/26-16:22:41.636288 scanner.5.43740 > home.44.1080: S 1124083233:1124083233(0) win 1024
2002/12/26-16:22:43.618842 scanner.5.44100 > home.44.3128: S 185666678:185666678(0) win 1024
2002/12/26-16:22:45.614627 scanner.5.29786 > home.44.8080: S 1231912456:1231912456(0) win 1024
2002/12/26-16:22:47.378890 scanner.5.29439 > home.44.80: S 780169159:780169159(0) win 1024
```

In addition to the port scans above, web server log files will sometimes show odd requests for other web servers, such as shown in an Apache web server's logs below.

```
10.2.2.214 - - [18/Nov/2002:13:19:00 -0500] "GET http://www.intel.com/ HTTP/1.1" 404 281 "-"
10.3.3.19 - - [06/Dec/2002:02:02:54 -0500] "GET http://www.yahoo.com/ HTTP/1.1" 404 281 "-"
10.2.2.214 - - [12/Dec/2002:02:49:11 -0500] "GET http://www.intel.com/ HTTP/1.1" 404 281 "-"
10.6.6.210 - - [13/Dec/2002:08:39:03 -0500] "HEAD http://www.sun .com" 400 - "-" "-"
10.4.4.176 - - [10/Jan/2003:08:28:00 -0500] "GET http://www.yahoo.com/ HTTP/1.1" 404 281 "-"
10.5.5.3 - - [14/Jan/2003:18:13:11 -0500] "GET http://www.google.com/ HTTP/1.1" 404 281 "-"
10.5.5.3 - - [14/Jan/2003:21:38:55 -0500] "GET http://www.google.com/ HTTP/1.1" 404 281 "-"
10.5.5.3 - - [15/Jan/2003:17:43:35 -0500] "GET http://www.google.com/ HTTP/1.1" 404 281 "-"
10.1.1.175 - - [22/Jan/2003:05:46:50 -0500] "GET http://www.intel.com/ HTTP/1.1" 404 281 "-"
```

These scans and logs are evidence of attackers searching for open HTTP proxy servers and are quickly becoming part of the background noise of the Internet, joining the likes of probes for NetBIOS shares and vulnerable IIS servers. Even though these scans may seem rather harmless, understanding what the scanners are looking for and what the they will do with any open proxy servers found is key to understanding how to better protect your network when proxy servers are present.

This paper will focus on how proxy servers are misconfigured and how they are misused. Tools currently used to scan for proxy servers and how these scans can be detected is also covered.

It should be pointed out that even though this paper focuses mainly on proxy servers that proxy web-based traffic, many of the problems associated with a misconfigured proxy server are universal and apply to all proxy servers. Additionally, all of the problems apply to proxy servers that are Internet accessible or segmented on a private network.

What is a proxy server?

The Merriam-Webster dictionary defines a proxy as “a person authorized to act for another”. Essentially, this is what a proxy server, sometimes referred to as an application gateway, does. A proxy server acts as an authorized server for a client by taking requests from the client and passing them to a foreign server. When the foreign server responds to the request, the proxy server will pass the response back to the original client.

A web browser uses an HTTP proxy server to contact foreign web servers on its behalf. The browser will send some HTTP commands, such as a GET request for the foreign server’s index.html page, to a pre-defined port on the proxy server. The proxy server will contact the foreign server with the client’s GET request and wait for the response. When the foreign server responds, the proxy server will send the data it received back to the original client. This communication will continue until the connection is closed.

Some of the HTTP proxy servers used today include the Microsoft Proxy Server, Squid, and the Wingate Proxy Server. Even some web servers, such as the Apache web server, can proxy some protocols. Each of these proxy servers listen on a different port for clients to send requests and because the default ports are well known, such as TCP port 3128 for the Squid proxy server, attackers know what ports to scan for.

However, proxy servers can only proxy protocols that they specifically know how to proxy, such as HTTP(S), TELNET or FTP. If any other protocol needs to be proxied, a SOCKS proxy server can be used.

The SOCKS protocol is used to proxy protocols that do not have a proxy server available. SOCKS is generic and does not control anything specific to any protocol, so it is very extensible (Zwicky 234). The latest version, SOCKS version 5, can proxy TCP and UDP protocols, provide user authentication and hostname resolution. SOCKS version 5 is defined in RFC 1928 and further described in RFC 1929, RFC 1951 and RFC 3089.

Most programs today are already SOCKS-aware and can use a SOCKS server without any modification. However, if the client program does not know how to communicate with a SOCKS server, it must be “socksified”. This is done in one of two ways. The first involves recompiling the source code of the program to include the SOCKS system calls that replace the normal network system calls. Since the source code of a program is not always available, a non-socksified program can use SOCKS dynamically by using a program such as SocksCap for Windows or runsocks for UNIX which will intercept any networking system calls from the program and convert them to the appropriate SOCKS system calls. Non-commercial versions of the SOCKS server, SocksCap and runsocks are freely available at http://www.socks.permeo.com.

Assuming a proxy server and it’s environment are set up correctly, there are many advantages to using a proxy server over letting individual clients contact foreign servers. First, since the proxy server handles all communication between the proxy server and a

foreign server, the foreign server never knows the true IP address of the client and cannot directly query them. This is useful, for example, when a client contacts a malicious web server that attempts to scan the client for vulnerable services. Instead of probing the client, the malicious server will scan the proxy server, never touching the original client.

A second advantage to using a proxy server is that the server can act as a choke point within the network. By adding authentication and access control on the proxy server and forcing clients to go through it, clients can be restricted to where they can go with each protocol proxied. Additionally, many proxy servers allow supplementary services, such as anti-virus scanning and content filtering, to be installed on the server which increases the security of the protocol being proxied and the network behind it.

Proxy Server Misconfigurations

However, these advantages will only work if the proxy server and network are set up correctly. If a proxy server or its surrounding network is misconfigured, serious security vulnerabilities can surface. Some common misconfigurations include lack of authentication and not properly protecting the proxy server.

Most, if not all, proxy servers provide some type of authentication to control who can use it. These methods vary according to the proxy server used and range from a username and password stored in a flat file to a backend LDAP server or database. Unfortunately, even though authentication is available, it is not always used. Problems arise when authentication is not used and accountability is needed to trace back who did what.

Proxy servers are often set up to proxy connections from an internal network to the Internet. When they are set up this way, they should be protected from the Internet in the same manner a host in a protected subnet would be. Protecting the proxy server in this way includes setting up a firewall to restrict connections to and from the proxy server for both its Internet and private network facing connections. Zwicky, Cooper and Chapman say it best in Building Internet Firewalls,

> Proxy Systems are effective only when they are used in conjunction with some method of restricting IP-level traffic between clients and the real servers, such as a screening router of a dual-homed host that doesn't route packets. If there is IP-level connectivity between the client and the real servers, the clients can bypass the proxy system (and presumably so can someone from the outside). (225)

Failure to effectively filter unwanted traffic to and from the proxy server, such as connections not initiated from the private network, may allow proxy servers to be used for malicious purposes or provide unprotected tunnels into the protected network.

Even though the misconfigurations described above are easy to fix and implement correctly, a surprisingly large number of proxy servers have these misconfigurations present and are considered open proxies on the Internet. An open proxy server is a proxy server that allows anyone to connect and use the proxy services for whatever they choose, although most open proxy servers only offer web-based or telnet connections. It is easy to find lists of proxy servers to use on the Internet and can be found for free at web sites such as http://void.ru and http://www.winfosec.com/proxies/.

How are open proxy servers misused?

In order to find out what open proxy servers are used for, I wrote a small open proxy honeypot Perl script similar to the script created by Joe Stewart in his article, “Exposing the Underground: Adventures of an Open Proxy Server”. The script acts as an open HTTP proxy server that will accept and log connections from anyone and behave differently based upon what the client requests.

Upon receiving an HTTP HEAD request, the script will return an “HTTP 200 OK” message to the client. If the request is an HTTP GET, the script will get the item requested from the foreign server and return it to the client as long as the item is not an image or the request contains an authentication attempt. If the request is for an image, a small, black GIF will be returned to the client. If the request contains an authentication attempt, the script returns a "401 Unauthorized" error message to simulate an invalid log in attempt. Finally, an HTTP CONNECT request will return a “404 Not Found” error message. The source code for the script is available in appendix A.

By running the script on my home machine for about 8 hours and advertising it as an open proxy server on a number of open proxy lists, I was able to get a number of connections to the honeypot and record what they did. There were four different things the open proxy server was used for: anonymous web surfing and proxy checking, brute force attacks against password protected web sites, proxying connections to mail servers and anonymous IRC connections.

The majority of the connections to the script were attempts to use it as an anonymous proxy server and surf the web. By using an anonymous proxy server, one can be assured that they can surf the web in relatively privacy, with little to no hope of anyone finding out their true IP address. This is useful, for example, when someone is posting to a message board and wishes to remain unidentified or when someone wishes to avoid the tracking mechanisms present on the Internet today. The sites that were connected to varied from adult sites to gaming discussion forums to eBay.

The following are some examples of attempts to connect to web sites with the script. The logs created by the proxy script show anything the client sent to the script. However, for brevity, some of the unnecessary information, such as the user agent and host header fields, have been stripped. The IP address of the originating request has also been sanitized.

192.168.1.3 GET http://runuo.com/ HTTP/1.0
192.168.1.3 GET http://www.runuo.com/styles/runuo.css HTTP/1.0
192.168.1.3 GET http://www.runuo.com/images/wmt-hilightshead.gif HTTP/1.0
192.168.1.3 GET http://www.runuo.com/discussion/viewtopic.php?t=1685&sid=d980796d09 HTTP/1.0
10.50.65.132 GET http://www.google.com/ HTTP/1.0
10.50.65.132 GET http://www.geocities.com/abyla47 HTTP/1.0
172.16.6.62 GET http://ebay.com/ HTTP/1.0
172.16.6.62 GET http://include.ebay.com/aw/pics/js/help/openHelpWindow.js HTTP/1.0
172.16.6.62 GET http://include.ebay.com/aw/pics/js/ebay_toolbar/installVB.js HTTP/1.0

Some of the web connections, however, were not to normal web sites but to scripts used to check proxy servers. These scripts will check if a proxy server is open or how anonymous a proxy server lets you become. The script shows the surfer how anonymous they are by displaying the headers that are returned from an HTTP request sent to the proxy server. The less information in the headers, the more anonymous the surfer is when using the proxy server. Usually, these scripts are hosted at the same sites that maintain lists of open proxy servers and whenever a proxy server is checked and found to be open, it will be added to the list.

The following are some examples of proxy checking scripts that were run against the proxy script.

172.16.182.124 GET http://www.gamox.com/cgi-bin/envinfo.cgi HTTP/1.0
10.1.14.2 GET http://badfellow.com/blood/pcheck/anonim.pl HTTP/1.1
192.168.122.194 GET http://void.ru/?do=envprox HTTP/1.0
10.2.65.250 GET http://atomintersoft.com/40.aspx?p=192.168.1.163:8080 HTTP/1.0

Since an open proxy server provides anonymous connections to web sites, it makes it a prime target to be used to relay brute force password attacks against password protected web sites. Brute force attacks attempt to log in to a web site using a large number of user ID and password combinations and recording which ones are successful. While a brute force attack may be launched against a site such as eBay or PayPal where a user ID and password could yield monetary gains, most of the brute force attacks witnessed were against adult oriented sites. In fact, there is a large community of password crackers dedicated to finding and sharing passwords to adult sites. Open proxy servers provide them the anonymity and protection they desire to launch brute force attacks against these web sites, as the web site will see the attack coming from the proxy server and not the attacker's true IP address.

The following is an attempt to use the proxy script to brute force a password on a website. Notice how quickly the attempts occur. In this particular case, 608 attempts occurred in a little under three minutes.

Mon, 25 Nov 2002 09:47:20 PM EST
10.10.216.189 HEAD http://www.picsofme.com/members/index.html HTTP/1.0
Authorization: Basic MTIzNDU6eHl6

Mon, 25 Nov 2002 09:47:21 PM EST
10.10.216.189 HEAD http://www.picsofme.com/members/index.html HTTP/1.0

Authorization: Basic c3dpZmZlcjpzd2VlcGVy

Mon, 25 Nov 2002 09:47:21 PM EST
10.10.216.189 HEAD http://www.picsofme.com/members/index.html HTTP/1.0
Authorization: Basic ZnVuZmFtbHk6bWF4ZWxs

Mon, 25 Nov 2002 09:47:21 PM EST
10.10.216.189 HEAD http://www.picsofme.com/members/index.html HTTP/1.0
Authorization: Basic Z2F5c2V4OmdheXNleA==

Mon, 25 Nov 2002 09:47:21 PM EST
10.10.216.189 HEAD http://www.picsofme.com/members/index.html HTTP/1.0
Authorization: Basic c3RldmUyMTpodXN0bGVy

Mon, 25 Nov 2002 09:47:21 PM EST
10.10.216.189 HEAD http://www.picsofme.com/members/index.html HTTP/1.0
Authorization: Basic cGFzc3dvcmQ6cGFzc3dvcmQ=

Mon, 25 Nov 2002 09:47:22 PM EST
10.10.216.189 HEAD http://www.picsofme.com/members/index.html HTTP/1.0
Authorization: Basic ZWJ1bWdhcmQ6dG95b3Rh

Not all of the connections to the proxy script were to web sites, some connections were attempts to connect to other TCP services through the script. When a proxy server wants to proxy an HTTPS connection, it cannot proxy the request the same way that it would an HTTP connection. If it did, the proxy server would have to decrypt the SSL encrypted data and pass it back to the client unencrypted, which would violate the privacy the encryption provides. In order to overcome this problem, Ari Luotonen described a method to tunnel TCP based protocols through HTTP proxy servers in his Internet Draft Paper.

In his paper, Luotonen describes the HTTP CONNECT method for proxy servers. This method creates a tunnel between the client and the foreign server through the proxy server. The client will connect to the proxy server as usual and send a CONNECT string specifying the server and port to contact as well as any other HTTP header information, such as authorization information. The proxy server will attempt to contact the foreign server on the port specified by the client. If the connection is successful, the proxy server will send the client a “200 Connection Successful” message and then data transfer between the client and the foreign server will occur. It should be noted that when the CONNECT method is used, the proxy server only forwards the traffic and does not examine any of it.

There is a problem with some proxy where they do not allow administrators to restrict what the destination port of a CONNECT tunnel can be. When the destination port is not restricted, a connection can be created to any TCP service and can provide another way for malicious users to get anonymous connections to services such as IRC, TELNET or SMTP. This vulnerability is noted in CERT Vulnerability Note #150227 and is briefly explained in Luotonen’s paper.

During the time the proxy script was running, attempts to use the CONNECT method to connect to TCP services other than HTTPS was seen multiple times. The majority of the attempts were to TCP port 25, SMTP, as shown below. Since SMTP headers record the IP address of the machine sending an email, open proxy servers permit spammers to connect to mail relays and hide their true IP address. This allows them to retain their ISP connections longer, as the complaints will flow back to the owners of the open proxy server and their ISP rather than the spammer and the spammer's ISP.

```
192.168.52.150 CONNECT nodrog.webmedia.pl:25 HTTP/1.0
192.168.52.150 CONNECT mail2.gofast.net:25 HTTP/1.0
192.168.52.150 CONNECT ns.caotus.ru:25 HTTP/1.0
192.168.52.150 CONNECT relay.ptc.spbu.ru:25 HTTP/1.0
192.168.52.150 CONNECT solace.me.uiuc.edu:25 HTTP/1.0
192.168.52.150 CONNECT mail.whatodo.ru:25 HTTP/1.0
192.168.52.150 CONNECT relay.wplus.net:25 HTTP/1.0
192.168.52.150 CONNECT mail.sto.telegate.se:25 HTTP/1.0
```

The next largest set of attempts to use the CONNECT method were to Internet Relay Chat (IRC) ports. IRC is often used by hackers to communicate with each other and share exploits. However, there are many attacks that can disconnect someone from an IRC session. These denial of service attacks are usually known as floods.

When an IRC user is able to connect to IRC through a proxy server and they are attacked, the proxy will get flooded instead of the user's real machine. This allows the user to connect back to the IRC server with ease. Furthermore, by connecting through a proxy server, the IRC user is anonymous, as their true IP address is not known.

The following log entries show attempted connections to IRC servers.

```
192.168.219.11 CONNECT irc.japsclan.com:6667 HTTP/1.0
10.58.169.218 CONNECT IRC.DATANET.EE:6667 HTTP/1.0
10.25.169.218 CONNECT irc.stealth.net:6667 HTTP/1.0
172.30.179.179 CONNECT irc.terra.es:6667 HTTP/1.0
172.26.100.54 CONNECT correos.islagrande.cu:6667 HTTP/1.0
10.125.255.105 CONNECT IRC.ELSITIO.COM:6667 HTTP/1.0
172.31.145.65 CONNECT irc.sbor.ru:6667 HTTP/1.0
```

The proxy script observed all of the uses of open proxy servers previously described. However, there is one use of an open proxy server that was not witnessed by the script - using an open proxy server to tunnel into an internal network. When a proxy server is set up, its purpose is usually to proxy connections from an internal network to the Internet. If the proxy server is not set up properly, there exists the chance that it could become a reverse proxy server and proxy connections from hosts on the Internet to hosts on the internal, protected network, bypassing any security in place.

This is how hacker Adrian Lamo was able to get into the internal networks of WorldCom, the New York Times and Excite@Home. According to several articles by Kevin Poulson, Lamo used misconfigured proxy servers at all of the companies sites to tunnel into the internal network and access such things as customer records, sensitive

employee information and even router maintenance tools. Using Lamo and the access that he gained using misconfigured proxy servers as an example, one can see how large a security risk a misconfigured proxy server can present.

Proxy Hunter

Port scanners such as nmap or SuperScan can be used to find servers with ports that proxy servers usually listen on, but these tools will only tell the attacker what ports are open and not if the server can be used as an open proxy. However, there are specialized programs that will scan for ports that open proxy servers listen on and test whether or not they can be used as an open proxy server. One of the more popular of these programs is called Proxy Hunter.

Proxy Hunter is a Windows program whose purpose is to scan a range of IP addresses looking for open proxy servers on ports specified by the user. The current version, 3.1 beta 1, is located on the Chinese web site http://dzc.126.com, but can be found in various other places around the Internet.

The program works by first having the user enter in the hosts to scan. The hosts can be entered in a number of ways, including single or ranges of IP addresses. After the hosts are entered, the user enters in the desired ports and services to scan for. Proxy Hunter comes with the ability to scan for HTTP, SOCKS, FTP, POP3 and TELNET proxy servers on any port; so a user could set the program to scan for an HTTP proxy on port 8080 or a TELNET proxy on port 12345. Proxy Hunter also comes with the ability to scan through existing open proxy servers, which provides the user protection and anonymity while they are scanning.

The following is an example of Proxy Hunter scanning a range of IP addresses for HTTP proxy servers on TCP ports 80 and 8080, and SOCKS servers on TCP port 1080. This is actual data from a scan, and was performed on my home network from a Windows XP Pro machine.

As shown below, Proxy Hunter will first scan for open ports on the IP range by sending SYN packets. The SYN packet it sends out is not unusual and there is no evidence of packet crafting. The only unusual item in the packet is all the TCP options that are set, but this was determined as normal after watching other traffic from the machine the tests were taking place on send the same options on SYN packets.

```
20:49:53.882917 IP 192.168.1.1.1356 > 192.168.1.10.8080: S
476313682:476313682(0) win 64512 <mss 1460,nop,wscale 0,nop,nop,timestamp 0
0,nop,nop,sackOK>
0x0000   4500 0040 1443 0000 8006 a319 c0a8 0101        E..@.C..........
0x0010   c0a8 010a 054c 1f90 1c63 f852 0000 0000        .....L...c.R....
0x0020   b002 fc00 2512 5800 0204 05b4 0103 0300        ....%.X.........
0x0030   0101 080a 0000 0000 0000 0000 0101 0402        ................
```

If Proxy Hunter does not receive a response from a SYN packet it will send a retry 3 and 6 seconds after the initial packet was sent. This is normal TCP behavior and again indicates packet crafting has not taken place. Typically when packet crafting has occurred, TCP retries do not follow the normal 3-6 second time frame.

So far, there has been nothing in Proxy Hunter's scan to differentiate between itself and a SYN scan from another tool. In order to figure out if Proxy Hunter is scanning a network, we will have to wait until it finds an open port to test.

Proxy Hunter tests each protocol differently, but when testing for HTTP proxy servers it will attempt to get one of four different URLs: www.click2net.com, www.intel.com, www.adm.com or www.spedia.net/sp_login.htm. This is shown in the packet below.

```
20:49:53.884966 IP 192.168.1.1.1356 > 192.168.1.10.8080: P 1:148(147) ack 1
win 64512
0x0000   4500 00bb 1445 0000 8006 a29c c0a8 0101        E....E.........
0x0010   c0a8 010a 054c 1f90 1c63 f853 c154 238a        .....L...c.S.T#.
0x0020   5018 fc00 afb4 0000 4745 5420 6874 7470        P.......GET.http
0x0030   3a2f 2f77 7777 2e61 646d 2e63 6f6d 2f20        ://www.adm.com/.
0x0040   4854 5450 2f31 2e31 0d0a 486f 7374 3a20        HTTP/1.1..Host:.
0x0050   7777 772e 6164 6d2e 636f 6d0d 0a41 6363        www.adm.com..Acc
0x0060   6570 743a 202a 2f2a 0d0a 5072 6167 6d61        ept:.*/*..Pragma
0x0070   3a20 6e6f 2d63 6163 6865 0d0a 5573 6572        :.no-cache..User
0x0080   2d41 6765 6e74 3a20 4d6f 7a69 6c6c 612f        -Agent:.Mozilla/
0x0090   342e 3020 2863 6f6d 7061 7469 626c 653b        4.0.(compatible;
0x00a0   204d 5349 4520 352e 3031 3b20 5769 6e64        .MSIE.5.01;.Wind
0x00b0   6f77 7320 3938 290d 0a0d 0a                    ows.98)....
```

Again, there is nothing out of the ordinary in the packet. In the HTTP request itself, there are no unusual settings either. The User-Agent setting, which tells the web server what browser is requesting the page, is set to "Mozilla/4.0" here. This is a configurable parameter within Proxy Hunter which allows the user to pick from a list of pre-defined user agents or define their own.

If the server responds with a "404 Not Found" error, Proxy Hunter will mark the result of that server as "undecidable" or "not matching". If the server does respond with a web page, Proxy Hunter will check it for a "Verification Data Parameter" in the returned page. The "Verification Data Parameter" is a string that verifies that the returned page is actually the one that Proxy Hunter wanted. For example, when Proxy Hunter is requesting www.intel.com, it will search for "Welcome to Intel" in the returned page. If it finds the "Verification Data Parameter", the IP address is marked as good and can be used as an open proxy server.

After Proxy Hunter finishes talking to a host, it will gracefully close the connection with a FIN/ACK packet. Once again, nothing unusual occurs.

```
20:49:55.483011 IP 192.168.1.10.8080 > 192.168.1.1.1356: F 933:933(0) ack 148
win 5840 (DF)
0x0000   4500 0028 721a 4000 4006 455a c0a8 010a        E..(r.@.@.EZ....
0x0010   c0a8 0101 1f90 054c c154 272e 1c63 f8e6        .......L.T'..c..
```

```
0x0020   5011 16d0 f2fe 0000 0000 0000 0000            P.............
```

Since nothing unusual occurs at the packet level when Proxy Hunter is scanning a network, how then can it be detected? It seems that there is no foolproof way to detect when Proxy Hunter is scanning your network. However, it is possible to tell if an attacker is scanning for open HTTP proxy servers.

Detecting Open Proxy Server Scans and Tests

There are a number of ways to detect open proxy server scans coming across a network. The following section will focus on using Snort to detect open HTTP proxy server scans and tests.

Snort comes with a number of signatures to detect open proxy server scans. In the scan.rules rule file, three signatures specifically will detect proxy scans.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 1080 (msg:"SCAN SOCKS Proxy
attempt"; flags:S; reference:url,help.undernet.org/proxyscan/;
classtype:attempted-recon; sid:615; rev:3;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 3128 (msg:"SCAN Squid Proxy
attempt"; flags:S; classtype:attempted-recon; sid:618; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 8080 (msg:"SCAN Proxy \(8080\)
attempt"; flags:S; classtype:attempted-recon; sid:620; rev:2;)
```

Each of these signatures will detect when a connection is attempted to TCP ports 1080, 3128 or 8080 on $HOME_NET, which is usually defined to be the IP address range Snort is watching. Additionally, if Snort's spp_portscan or spp_portscan2 preprocessors are turned on, they may detect scans for these ports.

Unfortunately, these signatures will only detect when attempts to contact those ports are made, not when an actual attempt to test a server as an open proxy occurs. Additionally, these signatures will not look at connections to TCP port 80, another common port that open proxy servers run on.

In order to detect attempts to find open proxy servers, new Snort signatures will have to be written. The following signatures will detect any attempt to test a server listening on any of the ports defined in the $HTTP_PORTS variable for an open proxy.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg: "Attempted HTTP
GET open proxy test"; flow: to_server, established; content: "GET
http\:\/\/"; nocase;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg: "Attempted HTTP
HEAD open proxy test"; flow: to_server, established; content: "HEAD
http\:\/\/"; nocase;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg: "Attempted HTTP
CONNECT open proxy test"; flow: to_server, established; content: "CONNECT ";
nocase; content: "http\/1\."; within: 64; nocase;)
```

The first two signatures watch the packet payload on an established connection to a server on an HTTP port for the strings “GET http://” or “HEAD http://”. The third signature watches the same port on an established connection for the string “CONNECT ” followed by “HTTP/1.”, within 64 bytes of the “CONNECT” string. These strings are what are seen when an attacker attempts to use a server as an open HTTP proxy server, as shown in the logs taken from the honeypot proxy script above.

Unfortunately, there may be some false positives with these signatures. For example, if Snort is watching a network where a legitimate proxy server is sitting, these signatures will alert constantly.

Additionally, these signatures are not foolproof. Since they are only looking for the strings above, if an attacker were to come up with a new way to test for open proxy servers, they would be able to bypass the signatures. However, these signatures will detect all of the proxy server scans the honeypot proxy script saw.

It was previously said that there is no way to detect Proxy Hunter from the packets that it sends out. However, it is possible to detect it if the attacker has not changed the default URLs to scan for. The following signatures will detect a possible Proxy Hunter scan on a web server by watching for HTTP GET requests for the default web sites Proxy Hunter attempts to connect to through the proxy.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg: "Proxy Hunter open
proxy scan attempt"; flow: to_server, established; content: "GET
http\:\/\/www.intel.com"; nocase;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg: "Proxy Hunter open
proxy scan attempt"; flow: to_server, established; content: "GET
http\:\/\/www.click2net.com"; nocase;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg: "Proxy Hunter open
proxy scan attempt"; flow: to_server, established; content: "GET
http\:\/\/www.adm.com"; nocase;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg: "Proxy Hunter open
proxy scan attempt"; flow: to_server, established; content: "GET
http\:\/\/www.spedia.net"; nocase;)
```

If these signatures are put to use with the previous signatures written, they should be placed before the previous ones or they will never get alerted on due to the way Snort processes rules.

References

IRC Hacking. 2 April 2003 <http://onlinesecurity.virtualave.net/hacking/irc.htm>.

IRC Warfare. 2 April 2003. <http://dooyoo-uk.tripod.com/mirc/floods.htm>.

Luotonen, Ari. "Tunneling TCP based protocols through Web proxy servers" 2 April 2003 <http://www.web-cache.com/Writings/Internet-Drafts/draft-luotonen-web-proxy-tunneling-01.txt>.

"Multiple vendors' HTTP proxy default configurations allow arbitrary TCP connections via HTTP CONNECT method." CERT. 2 April 2003 <http://www.kb.cert.org/vuls/id/150227>.

Poulsen, Kevin. "Lamo strikes again: WorldCom." The Register 12 June 2001. 2 April 2003 <http://www.theregister.co.uk/content/archive/23218.html>.

Poulsen, Kevin. "@Home's mis-configured proxy Excites hacker." The Register 29 May 2001. 2 April 2003 <http://www.theregister.co.uk/content/archive/19279.html>.

Poulsen, Kevin. "New York Times Internal Network Hacked." 26 February 2002. 2 April 2003 <http://online.securityfocus.com/news/342>.

SOCKS. 2 April 2003 <http://www.socks.permeo.com/>.

Stewart, Joe. "Exposing the Underground: Adventures of an Open Proxy Server ." 10 November 2002 <http://www.infosecwriters.com/texts.php?op=display&id=54>.

Zwicky, Elizabeth D., Simon Cooper, and D. Brent Chapman. Building Internet Firewalls. 2nd ed. Sepastopol: O'Reilly, 2000.

Appendix A – proxy.pl

```
#!/usr/bin/perl
#
# proxy.pl  by Tyler Hudak
#
# This perl script will listen on port 8080 and act as an HTTP open
# proxy server and do the following:
#
# - A HEAD request should return a 200 OK
# - A GET request should go out and get the page requested, except if
# it is a request that contains an authentication attempt or is for
# an image.  An authentication attempt will get a 401 Unauthorized
# message returned and an image will return the file pic.gif, which
# you must include!!!!!!
# - A CONNECT request will return a 404 Not Found error
# - anything else (including a POST) will get a 500 Internal Server
# error

use IO::Socket;
use FileHandle;
use LWP::UserAgent;
use Fcntl;

# set up socket
$main_sock = new IO::Socket::INET (LocalHost => 'localhost',
                        LocalPort => 8080,
                        Listen  => 5,
                        Proto   => 'tcp',
                        ReuseAddr  => 1,
                        );

die "Socket could not be created.  Reason: $!\n" unless ($main_sock);

# set up log file
open (LOGFILE, ">>log.txt") || die "Can't open log.txt: $!\n";

while ($new_sock = $main_sock->accept()) {

  $space = 0;

  $pid = fork();

  # Successful fork and I'm the child
  if ($pid == 0)
  {
    # get address of connection
    $hersockaddr    = getpeername($new_sock);
    ($port, $iaddr) = sockaddr_in($hersockaddr);
    $herstraddr     = inet_ntoa($iaddr);

    $date = `date +\"%a, %d %b %Y %X %Z\"`;
    print "$date got one!\n";

    # get first line and find out what type of request it is
```

```
$first = <$new_sock>;
$logout = $first;

# get rest of request
while ($space == 0)
{
  $input = <$new_sock>;

  @myword=split('',$input);

  # end of request
  if (ord($myword[0]) == 10 || ord($myword[1]) == 10) { $space++}
  $logout = $logout . $input;
}

# print out faked data to requestor

# if the request contains authorization return a 401 unauthorized
if ($first =~ /http\:\/\/\S+\:\S+\@\S+/ || $logout =~ /Authorization\:/)
{
  print $new_sock "HTTP/1.1 401 Unauthorized\n\n";
  print $new_sock "\nAuthorization Required\n";
}
# a HEAD request - print out a 200 OK message
elsif ($first =~ /head/i) {
  print $new_sock "HTTP/1.1 200 OK\n";
}
# else if its a connect, return a 404
elsif ($first =~ /connect/i) {
  print $new_sock "HTTP/1.1 404 Not Found\n";
}
elsif ($first =~ /\.gif|\.jpg/i) {
  print $new_sock "HTTP/1.0 200 OK\n";
  print $new_sock "Content-length: 807\n";
  print $new_sock "Content-type: image/gif\n\n";

  open (GIF, "pic.gif") || die "can't open gif: $!\n";
  binmode GIF;
  print $new_sock <GIF>;
  close GIF;
}
# else get the page and return it
else {
  $ua = LWP::UserAgent->new;

  if ($first =~ /get/i) {
    @request = split (" ",$first);
    $http_req = HTTP::Request->new(GET => $request[1]);
    $ua->timeout(7);
    $http_res = $ua->request($http_req);
    if ($http_res->is_success) {
      print $new_sock $http_res->content;
    } else {
      print $new_sock $http_res->error_as_HTML;
    }
  }
  else {
```

```
        print $new_sock "HTTP/1.1 500 Internal Server Error\n";
      }

    }

    $new_sock->shutdown(2);

    print LOGFILE "$date";
    print LOGFILE "$herstraddr\n";
    print LOGFILE $logout;

    exit 0;
  } # child
  # i'm the parent - wait for another connection
  elsif ($pid > 0) {
    wait();
 }
  else {
    die "Error while forking: $!\n";
    exit;
  }

} # while

close $main_sock;
close LOGFILE;
```

Network Detects

Detect #1 - ATTACK RESPONSES id check returned root

Alert generated:

```
[**] [1:498:3] ATTACK RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
05/31-08:12:19.804488 205.252.49.1:80 -> 226.185.106.176:64094
TCP TTL:117 TOS:0x0 ID:38245 IpLen:20 DgmLen:1500 DF
***AP*** Seq: 0x6E1E09D0  Ack: 0x13A13701  Win: 0x43BC  TcpLen: 20
```

Actual packet that generated the alert:

```
$ tcpdump -r 2002.4.31.log -n -v 'src host 205.252.49.1 and dst host
226.185.106.176 and dst port 64094'

08:12:19.804488 205.252.49.1.80 > 226.185.106.176.64094: P [bad tcp cksum
8f8f!] 1847462352:1847463812(1460) ack 329332481 win 17340 (DF) (ttl 117, id
38245, len 1500, bad cksum 8ebf!)
```

1. Source of Trace:

The alert generated was obtained from the raw tcpdump log file generated by Snort and located at http://www.incidents.org/logs/Raw/2002.4.31. The file was sanitized as outlined in http://www.incidents.org/logs/Raw/README.

While I do not have access to the network layout, I can make an educated guess. The following was derived from my own analysis and Andre Cormier's excellent Network Detect submission.

Every packet in the tcpdump log is either coming from or going to an address within the 226.185.0.0/16 range. However, the log only shows a source IP address range of 226.185.106.0/24. This means that even though the entire IP address range of 226.185.0.0/16 is owned by this network, only 226.185.106.0/26 has hosts on it or only has hosts that Snort generated alerts on.

Additionally, there are only two MAC addresses for all packets in the log. According to IEEE these MAC addresses, 00:00:0c:04:b2:33 and 00:03:e3:d9:26:c0, are owned by Cisco cards. The MAC address 00:03:e3:d9:26:c0 is always associated with an external IP address and the MAC address 00:00:0c:04:b2:33 is always associated with an internal IP address in the tcpdump log. With this information, the following diagram on how the network is set up, at least in relation to the Snort IDS, can be generated:

```
Internet <-> Cisco Device  <-|-> Cisco Device <-> Internal Network
             0:3:e3:d9:26:c0 |   0:0:c:4:b2:33    (216.185.0.0/26)
                             |
                           Snort
```

Traffic coming from the Internet first goes through a Cisco device, probably a router or PIX firewall. The traffic is then detected by the Snort IDS sensor, most likely through a hub or a span port on a switch, and then goes to a second Cisco device. The second device is likely another PIX firewall or router that filters the data further and sends it off to the internal network.

2. Detect was generated by:

This detect was generated by running Snort Version 1.9.0 Build 209 against the tcpdump log specified above. A default snort.conf configuration file was used with an updated rule set. Snort was run to log to the current directory and to include packets in tcpdump format using the following options:

```
$ snort –b –r 2002.4.31 –c ./snort.conf –l ./
```

Snort is an open source network intrusion detection system created and maintained by Marty Roesch. It can be found at http://www.snort.org.

The rule that set off this alert is shown below:

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

We can analyze the signature to determine why this detect was generated.

```
alert ip any any -> any any
```

The first part of the signature, known as the rule header, tells Snort what to look for in regards to who is involved in the network conversation. In this case, the rule tells Snort to look for IP traffic coming from any IP address on any port going to any IP address on any port. In other words, this rule will get evaluated on any IP traffic whatsoever.

```
(msg:"ATTACK RESPONSES id check returned root";
```

Next we look at the rule options of the signature, located in the parentheses after the rule header. This tells Snort what parts within the packet it should look at as well as what output options to use. The first part, shown above, is the alert message to display when this alert is generated.

```
content: "uid=0(root)";
```

The content option tells Snort what to look for within the data of the packet. The rule here looks for the string “uid=0(root)”, which is part of the output from the UNIX “id” command. If Snort finds that string within the packet, the alert is generated.

```
classtype:bad-unknown; sid:498; rev:3;)
```

The rest of the signature gives event information on this alert to Snort. The classtype option defines this alert as “bad-unknown”, sid gives a signature identifier to the alert and rev is the revision of the alert.

Since the only option that looks into the packet payload is the content option, this alert will be generated whenever the string “uid=0(root)” is in an IP packet coming from and going to any IP address and port.

3. Probability the source address was spoofed:

There is little probability that the source address was spoofed. As will later be explained, the traffic that generated the alert appears to be part of a KaZaa file transfer. A file transfer using the KaZaa peer to peer file sharing agent occurs when a KaZaa client, internal host 226.185.106.176 in this case, contacts another KaZaa client (acting as a server) to download a file.

Since the internal host contacted the external host to download a file, it initiated the connection. While not impossible, it would be extremely difficult for another machine to intercept the traffic destined for 205.242.49.1 and hijack the connection. If we look further into the tcpdump logs, more connections between the two hosts take place. Since all of the connections seem to occur without any problems, the likelihood that the source was spoofed is very small.

The source address would probably not be spoofed even if this alert were generated from an actual attack and not a false positive. As will be described below, this alert is commonly generated after a machine has been compromised and output from the “id” command is sent back to the attacker. If the source address was spoofed, the attacker may never see the output and would not know that they had superuser access to the machine.

4. Description of attack:

When run, the UNIX “id” command displays the real and effective user IDs (UID) and group IDs (GID) for the current user. This alert is generated when the output from the “id” command shows that the user has the UID of 0. When a user has a UID of 0, they have superuser, or root, access to the machine. Attackers usually run the “id” command after they have compromised a machine to verify they have superuser access over the box, so this alert usually lets an administrator know that they have already been compromised.

In this case, the alert is a false positive. This is better seen if we use tcpdump to look at the actual data in the packet that generated the alert. Since there are multiple packets to and from these hosts, we will include the destination port in the BPF filter to get only the packet that generated the alert.

```
$  tcpdump -r 2002.4.31 -n -s 1514 -X 'src host 205.252.49.1 and dst host
226.185.106.176 and dst port 64094'

08:12:19.804488 205.252.49.1.80 > 226.185.106.176.64094: P
1847462352:1847463812(1460) ack 329332481 win 17340 (DF)

0x0000   4500 05dc 9565 4000 7506 8ebf cdfc 3101     E....e@.u.....1.
0x0010   e2b9 6ab0 0050 fa5e 6e1e 09d0 13a1 3701     ..j..P.^n.....7.
0x0020   5018 43bc a093 0000 6c63 6f64 653e 5c70     P.C.....lcode>\p
0x0030   6172 0d0a 7b5c 706e 7465 7874 5c66 315c     ar..{\pntext\f1\
0x0040   2742 375c 7461 627d 3233 302d 4e65 7874     'B7\tab}230-Next
0x0050   2074 696d 6520 706c 6561 7365 2075 7365     .time.please.use
0x0060   2079 6f75 7220 652d 6d61 696c 2061 6464     .your.e-mail.add
0x0070   7265 7373 2061 7320 796f 7572 2070 6173     ress.as.your.pas
0x0080   7377 6f72 645c 7061 720d 0a7b 5c70 6e74     sword\par..{\pnt
0x0090   6578 745c 6631 5c27 4237 5c74 6162 7d32     ext\f1\'B7\tab}2
0x00a0   3330 2d20 2020 2020 2020 2066 6f72 2065     30-........for.e
0x00b0   7861 6d70 6c65 3a20 6a6f 6540 6b67 622e     xample:.joe@kgb.
0x00c0   7a61 2e6e 6574 5c70 6172 0d0a 7b5c 706e     za.net\par..{\pn
0x00d0   7465 7874 5c66 315c 2742 375c 7461 627d     text\f1\'B7\tab}
0x00e0   3233 3020 4775 6573 7420 6c6f 6769 6e20     230.Guest.login.
0x00f0   6f6b 2c20 6163 6365 7373 2072 6573 7472     ok,.access.restr
0x0100   6963 7469 6f6e 7320 6170 706c 792e 5c70     ictions.apply.\p
... (cut for brevity) …
x02d0   4c69 6e75 7820 6c61 6d65 5f62 6f78 2e7a      Linux.lame_box.z
0x02e0   612e 6e65 7420 322e 322e 3134 2d35 2e30     a.net.2.2.14-5.0
0x02f0   2023 3120 5475 6520 4d61 7220 3720 3231     .#1.Tue.Mar.7.21
0x0300   3a30 373a 3339 2045 5354 2032 3030 3020     :07:39.EST.2000.
0x0310   6936 3836 2075 6e6b 6e6f 776e 5c70 6172     i686.unknown\par
0x0320   0d0a 7b5c 706e 7465 7874 5c66 315c 2742     ..{\pntext\f1\'B
0x0330   375c 7461 627d 7569 643d 3028 726f 6f74     7\tab}uid=0(root
0x0340   2920 6769 643d 3028 726f 6f74 2920 6567     ).gid=0(root).eg
0x0350   6964 3d35 3028 6674 7029 2067 726f 7570     id=50(ftp).group
0x0360   733d 3530 2866 7470 295c 7061 720d 0a7b     s=50(ftp)\par..{
0x0370   5c70 6e74 6578 745c 6631 5c27 4237 5c74     \pntext\f1\'B7\t
0x0380   6162 7d5c 7061 720d 0a7b 5c70 6e74 6578     ab}\par..{\pntex
0x0390   745c 6631 5c27 4237 5c74 6162 7d42 616e     t\f1\'B7\tab}Ban
0x03a0   6721 2059 6f75 2068 6176 6520 726f 6f74     g!.You.have.root
0x03b0   215c 7061 720d 0a7b 5c70 6e74 6578 745c     !\par..{\pntext\
0x03c0   6631 5c27 4237 5c74 6162 7d2d 2d2d 2d2d     f1\'B7\tab}-----
0x03d0   2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d     ----------------
0x03e0   2d2d 2d2d 2d2d 2d2d 2063 7574 2068 6572     --------.cut.her
0x03f0   6520 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d     e.--------------
0x0400   2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d5c     ---------------\
0x0410   7061 720d 0a7b 5c70 6e74 6578 745c 6631     par..{\pntext\f1
0x0420   5c27 4237 5c74 6162 7d54 6861 7473 2069     \'B7\tab}Thats.i
0x0430   742e 2e2e 2077 6861 7420 796f 7520 646f     t....what.you.do
0x0440   2066 726f 6d20 6865 7265 2069 7320 7468     .from.here.is.th
0x0450   6520 6d61 7474 6572 206f 6620 6f74 6865     e.matter.of.othe
0x0460   7220 686f 7732 732e 2059 6f75 205c 7061     r.how2s..You.\pa
0x0470   720d 0a7b 5c70 6e74 6578 745c 6631 5c27     r..{\pntext\f1\'
0x0480   4237 5c74 6162 7d61 6c73 6f20 6d69 6768     B7\tab}also.migh
0x0490   7420 6265 2061 736b 696e 6720 7768 6174     t.be.asking.what
0x04a0   2069 7320 4e65 7443 6174 2066 6f72 2e2e     .is.NetCat.for..
0x04b0   2e20 7765 6c6c 2073 6f6d 6520 6578 706c     ..well.some.expl
```

```
0x04c0   6f69 7473 2072 6571 7569 7265 205c 7061     oits.require.\pa
0x04d0   720d 0a7b 5c70 6e74 6578 745c 6631 5c27     r..{\pntext\f1\'
0x04e0   4237 5c74 6162 7d69 742e 204e 6f74 6963     B7\tab}it..Notic
0x04f0   6520 7468 6174 2061 626f 7665 2065 7870     e.that.above.exp
0x0500   6c6f 6974 2075 7365 6420 616e 6f6e 796d     loit.used.anonym
0x0510   6f75 7320 6c6f 6769 6e2c 2073 6f20 6966     ous.login,.so.if
0x0520   2061 6e6f 6e79 6d6f 7573 205c 7061 720d     .anonymous.\par.
0x0530   0a7b 5c70 6e74 6578 745c 6631 5c27 4237     .{\pntext\f1\'B7
0x0540   5c74 6162 7d61 6363 6573 7320 7761 7320     \tab}access.was.
0x0550   6469 7361 626c 6564 2074 6865 7265 2c20     disabled.there,.
0x0560   6974 2077 6f75 6c64 6e74 2077 6f72 6b2e     it.wouldnt.work.
0x0570   2054 6861 7473 2077 6879 2077 6520 7765     .Thats.why.we.we
0x0580   7265 2063 6865 636b 696e 675c 7061 720d     re.checking\par.
0x0590   0a7b 5c70 6e74 6578 745c 6631 5c27 4237     .{\pntext\f1\'B7
0x05a0   5c74 6162 7d66 6f72 2061 6e6f 6e79 6d6f     \tab}for.anonymo
0x05b0   7573 2061 6363 6573 7320 6174 2073 7465     us.access.at.ste
0x05c0   7020 662e 2049 6620 616e 6f6e 2061 6363     p.f..If.anon.acc
0x05d0   6573 7320 7761 7320 6469 7361               ess.was.disa
```

Looking at the data in the packet, we see the string “uid=0(root)”, which set off the alert, surrounded by what looks like an RTF document. The RTF document looks strangely like a cracking tutorial. In fact, if we search the Internet using Google for the phrase “Bang! You have root!” we find a tutorial by kgb_kid called “Cracking Howto 1” which explains how to run a wu-ftp exploit.

Looking further into the tcpdump logs for any other packets between these two hosts, we come across the following packet:

```
$ tcpdump -r 2002.4.31 -n -s 1514 -X 'src host 205.252.49.1 and dst host
226.185.106.176’

… (some output cut for brevity) …
13:58:52.704488 205.252.49.1.80 > 226.185.106.176.61188: P
1462836325:1462837785 (1460) ack 2582223155 win 17340 (DF)

0x0000   4500 05dc 1a47 4000 7506 09de cdfc 3101     E....G@.u.....1.
0x0010   e2b9 6ab0 0050 ef04 5731 1c65 99e9 9933     ..j..P..W1.e...3
0x0020   5018 43bc 8ef3 0000 4854 5450 2f31 2e31     P.C.....HTTP/1.1
0x0030   2032 3030 204f 4b0d 0a43 6f6e 7465 6e74     .200.OK..Content
0x0040   2d4c 656e 6774 683a 2033 3339 3534 0d0a     -Length:.33954..
0x0050   4163 6365 7074 2d52 616e 6765 733a 2062     Accept-Ranges:.b
0x0060   7974 6573 0d0a 4461 7465 3a20 4672 692c     ytes..Date:.Fri,
0x0070   2033 3120 4d61 7920 3230 3032 2031 393a     .31.May.2002.19:
0x0080   3538 3a35 3820 474d 540d 0a53 6572 7665     58:58.GMT..Serve
0x0090   723a 204b 617a 6161 436c 6965 6e74 204d     r:.KazaaClient.M
0x00a0   6179 2032 3820 3230 3032 2030 303a 3233     ay.28.2002.00:23
0x00b0   3a35 320d 0a43 6f6e 6e65 6374 696f 6e3a     :52..Connection:
… (cut for brevity) …
```

This is the start of an HTTP server response from the external host. Examining the HTTP Server header more closely, we see that it is a KaZaa client as shown by the HTTP header “Server: KazaaClient” highlighted above. KaZaa is a peer to peer file-sharing network where users utilize the KaZaa agent to download any files shared by

any other user. Users who have the KaZaa client installed are able to change the port it listens for requests on, from the default port of TCP 1214 to whatever port they want. This is done to more easily bypass firewalls, since a firewall is more likely to allow inbound or outbound traffic to a port such as TCP 80 than TCP 1214.

What appears to be happening here is that a user on the network is using KaZaa to download tutorials on how to hack computers. The tutorial shows the output of one particular exploit, which happens to be the output of the "id" command from a user who has successfully gained root access on a Linux machine using a remote FTP exploit. Since this output contains the string "uid=0(root)", it set off the signature in Snort, creating the false positive.

5. Attack mechanism:

Many exploits that attack systems do so remotely. Therefore, any traffic, including any output, from the exploit has to travel over the network back to the attacker. Many of these remote exploits run the "id" command automatically after it has successfully compromised a machine to verify to the attacker that they have superuser, or root, access. The "ATTACK RESPONSES id check returned root" signature exists to detect this.

While we would hope that Snort would generate an alert on the actual exploit attacking the machine, this may not happen, especially if the attack was unknown. This signature provides a way to alert in case Snort fails to detect the attack.

6. Correlations:

The Snort log shows 24 other alerts set off by the communications between these two hosts. Every one of these alerts is either a "SHELLCODE x86 inc ebx NOOP" or "SHELLCODE x86 NOOP" alert. Each of these alerts occurs when an exploit is being run against a remote machine. However, these alerts can also occur when someone is downloading an exploit containing the shellcode signature that Snort looks for. Considering the previous evidence that the user is using KaZaa to download cracking tutorials, chances are they are also using KaZaa to download the exploits mentioned within these tutorials. These exploits would be what Snort is detecting.

There are many instances where the "ATTACK RESPONSES id check returned root" alert is a false positive. For example, in his GIAC network detect, Dennis Klaman reported a similar false positive with the alert.

However, many incident response reports available on the Internet show the signature occurring after a machine compromise has occurred. One such incident report is Michael Anuzis' analysis of a compromised BSD honeypot.

As per guidelines, this detect was posted on the Intrusions mailing list on January 31, 2003. There were no responses to this so it was resubmitted to the list again on March

8, 2003. Andrew Rucker Jones replied to it and said that the analysis was good and that he had nothing to say about it.
Andrew also replied with the following:

> The Snort log shows 24 other alerts set off by the communications between
> these two hosts. Every one of these alerts is either a "SHELLCODE x86 inc ebx
> NOOP" or "SHELLCODE x86 NOOP" alert. Each of these alerts occurs when an
> exploit is being run against a remote machine. However, these alerts can
> also occur when someone is downloading an exploit containing the shellcode
> signature that Snort looks for.

Not only. This is a frequent false alarm. Downloading other files that KaZaa is associated with (like MP3's) would probably generate a few of these alerts, too. But, as You say, the alerts are between those two hosts, so it stands to reason...

I agree with Andrew's comment here that downloading other files can set off these alerts, but given that there is evidence the user is downloading hacking tutorials, I believe that these alerts were set off due to exploits being downloaded.

7. Evidence of active targeting:

KaZaa, and other file sharing software, work by letting different clients connect to each other to download software. In this case, the internal user actively contacted the remote machine to download the file. There is no evidence that the external machine targeted the internal machine in any way.

8. Severity:

The equation for severity is:

severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Each value is rated on a scale from 1 to 5, where 1 is the lowest value.

Criticality – We don't know how critical the system is, but all of our conclusions have pointed to it being a user's system or workstation. Since this is on the lower end of critical machines as compared to web and DNS servers, criticality will be given a value of 2.

Lethality – The potential for a positive "id check returned root" alert gives it the highest lethality possible since the alert would indicate a root compromise of a machine. This alert is a false positive, though, and should get a low rating. However, since the data within the packet indicates some other nefarious activities may be planned and exploits may have been downloaded, this category should be rated higher. Lethality will be given a rating of 2.

System Countermeasures – We do not know what system countermeasures are available on the host system, so it gets a low score of 1.

Network Countermeasures – While we do not know exactly what network countermeasures are in place, we do know that some type of Cisco border router or firewall and a Snort IDS is in place. This would normally lead to a high value, but since the border protection let the peer to peer file sharing through, it should be lessened a little. Network Countermeasures will get a value of 2.

Severity = (2+2) – (1+2) = 4 – 3 = 1

A severity of 1 indicates some additional network countermeasures, as specified below, need to be put into place. Follow-up with the user associated with this incident should occur as well.

9. Defensive recommendation:

Normally there are no defensive recommendations for this alert, except taking the appropriate steps to prevent whatever compromise had occurred from happening again. This would include blocking ports on a firewall, applying patches and end user education.

Since this alert was a false positive and no compromise actually occurred, there are no defensive recommendations. However, it would be prudent to contact the internal user downloading these files as it is obvious they are trying to learn how to hack and may try to apply that knowledge against the other machines on the network.

However, it may also be helpful to block internal users from using a peer to peer file sharing program like KaZaa to prevent viruses and other malware from entering the network. Blocking peer to peer file-sharing programs can also prevent potential legal liability from users downloading copyrighted materials, such as MP3s. Blocking can be accomplished in a number of ways.

The first step is usually blocking the ports used by peer to peer programs. For the KaZaa peer to peer network, the port to block would be TCP 1214. However, as seen in this alert, many peer to peer programs now have the ability to change the port they listen on for the express purpose of bypassing firewall restrictions.

Since most peer to peer file-sharing programs require you to log on to a central server, blocking the IP address ranges of these servers may stop some programs from working. This is because if the client is not able to authenticate onto the peer to peer network, they won't be able to use it. The list of central servers used often changes however, so the list of blocked IP addresses would need to be updated frequently.

The next best thing to do would be to require all users to go through an authenticating proxy server to get to the Internet. Some of the peer to peer file-sharing programs lack the ability to work through a proxy server. The ones that are able to work through a proxy server do not always have the ability to work through one that requires

authentication. While not foolproof, using the proxy server will help prevent some of the peer to peer file-sharing programs from working.

10. Multiple choice test question:

```
13:58:52.704488 205.252.49.1.80 > 226.185.106.176.61188: P
1462836325:1462837785 (1460) ack 2582223155 win 17340 (DF)

0x0000   4500 05dc 1a47 4000 7506 09de cdfc 3101     E....G@.u.....1.
0x0010   e2b9 6ab0 0050 ef04 5731 1c65 99e9 9933     ..j..P..W1.e...3
0x0020   5018 43bc 8ef3 0000 4854 5450 2f31 2e31     P.C.....HTTP/1.1
0x0030   2032 3030 204f 4b0d 0a43 6f6e 7465 6e74     .200.OK..Content
0x0040   2d4c 656e 6774 683a 2033 3339 3534 0d0a     -Length:.33954..
0x0050   4163 6365 7074 2d52 616e 6765 733a 2062     Accept-Ranges:.b
0x0060   7974 6573 0d0a 4461 7465 3a20 4672 692c     ytes..Date:.Fri,
0x0070   2033 3120 4d61 7920 3230 3032 2031 393a     .31.May.2002.19:
0x0080   3538 3a35 3820 474d 540d 0a53 6572 7665     58:58.GMT..Serve
0x0090   723a 204b 617a 6161 436c 6965 6e74 204d     r:.KazaaClient.M
0x00a0   6179 2032 3820 3230 3032 2030 303a 3233     ay.28.2002.00:23
0x00b0   3a35 320d 0a43 6f6e 6e65 6374 696f 6e3a     :52..Connection:
… (cut for brevity) …
```

If 226.185.106.176 is on the internal network, what activity is the above packet accomplishing?

a) Bypassing firewall restrictions
b) Connecting to a backdoor
c) Sending a buffer overflow
d) Nothing, this is normal HTTP traffic

Answer: A. The port of the service being used, KaZaa in this case, is listening on port 80. Most firewalls allow port 80, commonly used by HTTP, through to allow users to surf the web. By listening on port 80, the user is almost guaranteed to be allowed to use KaZaa through the firewall, something that may normally be restricted in the firewall policy.

11. References:

Anuzis, Michael. "Incident Analysis of Compromised OpenBSD 3.0 Honeypot." July 2002. <http://www.lucidic.net/whitepapers/manuzis-7-5-2002-1.html>.

Cormier, Andre. "LOGS: GIAC GCIA Version 3.3 Practical Detect(s) (Andre Cormier)." 20 Jan 2003. Intrusions mailing list. 24 Jan 2003. <http://cert.uni-tuttgart.de/archive/intrusions/2003/01/msg00162.html>.

"Google search: kgb_kid." 24 Jan. 2003. <http://www.google.com/search?q=%22kgb_kid%22&hl=en&lr=&ie=UTF-8&oe=UTF-8>.

Hudak, Tyler. "LOGS: GIAC GCIA Version 3.3 Practical Detect (Hudak)." 31 Jan 2003. Intrusions mailing list. 2 April 2003. <http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00510.html>.

Hudak, Tyler. "LOGS: GIAC GCIA Version 3.3 Practical Detect (Hudak) – second posting." 8 Mar 2003. Intrusions mailing list. 2 April 2003. <http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00105.html>.

Jones, Andrew Rucker. "Re: LOGS: GIAC GCIA Version 3.3 Practical Detect (Hudak) – second posting." 8 Mar 2003. Intrusions mailing list. 2 April 2003. <http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00111.html>.

KGB_Kid. "Cracking Howto 1." 10 May 2000. Google cache. <http://216.239.39.100/search?q=cache:S2XrOR5SD58C:bigben187.housing4free.de/docs/basic_hacking_guides_using_linux_tru_WUftpd_bughole.rtf+%22kgb_kid%22&hl=en&ie=UTF-8>.

Klaman, Dennis. "LOGS: GIAC GCIA Version 3.3 Detect #1 (Klaman)." Online posting. 21 Oct 2002. Intrusions mailing list. 24 Jan 2003. <http://cert.uni-tuttgart.de/archive/intrusions/2002/10/msg00266.html>.

Detect #2 - BAD TRAFFIC udp port 0 traffic

Alerts generated:

```
[**] [1:525:4] BAD TRAFFIC udp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
02/04-18:47:23.860953 208.203.77.22:53 -> 192.168.25.15:0
UDP TTL:1 TOS:0x0 ID:27417 IpLen:20 DgmLen:64
Len: 44
[Xref => nessus 10074][Xref => cve CVE-1999-0675]

[**] [1:525:4] BAD TRAFFIC udp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
02/04-18:47:24.866417 208.203.77.22:53 -> 192.168.25.15:0
UDP TTL:1 TOS:0x0 ID:27418 IpLen:20 DgmLen:64
Len: 44
[Xref => nessus 10074][Xref => cve CVE-1999-0675]

[**] [1:525:4] BAD TRAFFIC udp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
02/04-18:47:25.876605 208.203.77.22:53 -> 192.168.25.15:0
UDP TTL:2 TOS:0x0 ID:27419 IpLen:20 DgmLen:64
Len: 44
[Xref => nessus 10074][Xref => cve CVE-1999-0675]
[Xref => nessus 10074][Xref => cve CVE-1999-0675]

[**] [1:525:4] BAD TRAFFIC udp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
02/04-18:47:26.886692 208.203.77.22:53 -> 192.168.25.15:0
UDP TTL:2 TOS:0x0 ID:27420 IpLen:20 DgmLen:64
Len: 44
[Xref => nessus 10074][Xref => cve CVE-1999-0675]

[**] [1:525:4] BAD TRAFFIC udp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
02/04-18:47:27.896698 208.203.77.22:53 -> 192.168.25.15:0
UDP TTL:2 TOS:0x0 ID:27421 IpLen:20 DgmLen:64
Len: 44
[Xref => nessus 10074][Xref => cve CVE-1999-0675]
```

Traffic that generated the alerts:

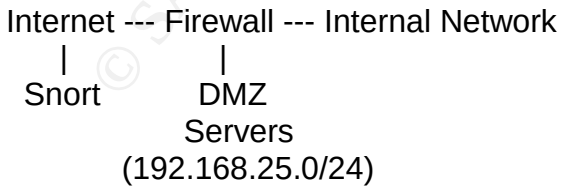
```
$ tcpdump -r snort.log.1042744510 -n -X -s 1514 -q 'udp and src host
208.203.77.22 and dst port 0'

18:47:23.860953 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36 [ttl
1] (id 27417, len 64)
0x0000       4500 0040 6b19 0000 0111 e84f d0cb 4d16  E..@k......O..M.
0x0010       c0a8 190f 0035 0000 002c ad81 6b19 8081  .S...5...,..k...
0x0020       0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030       0000 0000 0000 0000 0000 0000 0000 0000  ................
```

```
18:47:24.866417 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36 [ttl
1] (id 27418, len 64)
0x0000       4500 0040 6b1a 0000 0111 e84e d0cb 4d16  E..@k......N..M.
0x0010       c0a8 190f 0035 0000 002c ad80 6b1a 8081  .S...5...,..k...
0x0020       0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030       0000 0000 0000 0000 0000 0000 0000 0000  ................

18:47:25.876605 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36 (ttl
2, id 27419, len 64)
0x0000       4500 0040 6b1b 0000 0211 e74d d0cb 4d16  E..@k.....M..M.
0x0010       c0a8 190f 0035 0000 002c ad7f 6b1b 8081  .S...5...,..k...
0x0020       0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030       0000 0000 0000 0000 0000 0000 0000 0000  ................

18:47:26.886692 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36 (ttl
2, id 27420, len 64)
0x0000       4500 0040 6b1c 0000 0211 e74c d0cb 4d16  E..@k......L..M.
0x0010       c0a8 190f 0035 0000 002c ad7e 6b1c 8081  .S...5...,.~k...
0x0020       0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030       0000 0000 0000 0000 0000 0000 0000 0000  ................

18:47:27.896698 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36 (ttl
2, id 27421, len 64)
0x0000       4500 0040 6b1d 0000 0211 e74b d0cb 4d16  E..@k......K..M.
0x0010       c0a8 190f 0035 0000 002c ad7d 6b1d 8081  .S...5...,.}k...
0x0020       0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030       0000 0000 0000 0000 0000 0000 0000 0000  ................
```

1. Source of Trace:

This detect was obtained from a Snort IDS listening to my employer's Internet connection.

Without going too deeply into the structure of the network, the IDS listens to all Internet traffic outside of the firewall through an interface in stealth mode. This means the IDS sensor's interface has no IP address assigned to it so the interface can only listen to traffic and not respond. Behind the firewall reside the protected e-commerce servers and internal network. The following is a basic diagram of how the network is organized.



It should be noted that in this alert the destination IP address, 192.168.25.15, is obfuscated to hide the actual class C Internet IP address and internal IP address of the machine in the alert. The source IP address, 208.203.77.22, has not been changed.

2. Detect was generated by:

The detect was generated by Snort Version 1.9.0 Build 209 running a rule set current as of February 3, 2003. Snort is an open source Intrusion Detection System maintained by Marty Roesch and located at http://www.snort.org.

The signature that set off the "BAD TRAFFIC udp port 0 traffic" alert is shown below:

```
alert udp $EXTERNAL_NET any <> $HOME_NET 0 (msg:"BAD TRAFFIC udp port 0
traffic"; reference:cve,CVE-1999-0675; reference:nessus,10074;
classtype:misc-activity; sid:525; rev:4;)
```

We can analyze the signature to determine why the alert was generated.

```
alert udp $EXTERNAL_NET any <> $HOME_NET 0
```

The rule header of the signature tells Snort that this signature should only look at UDP network traffic coming from or going to port 0 on any machine in $HOME_NET.

$HOME_NET is an internal variable set in Snort's configuration file that tells Snort what its home network should be. The home network is typically the IP address or range of the machine or network that Snort is watching. In this case, $HOME_NET was set to the class C IP range of the Internet connection it was listening on, or 192.168.25.0/24.

```
(msg:"BAD TRAFFIC udp port 0 traffic";
```

The first rule option in the signature shows the alert message Snort should display when this signature goes off.

```
reference:cve,CVE-1999-0675; reference:nessus,10074; classtype:misc-
activity; sid:525; rev:4;)
```

The rest of the signature provides Snort references and identification for the rule. These options have no bearing on when Snort will alert due to this signature.

Since there are no rule options that look into the packet, Snort will generate this alert whenever UDP traffic is seen coming from or going to port 0 on a machine in $HOME_NET.

3. Probability the source address was spoofed:

There is a possibility that the source address was spoofed. As opposed to TCP, UDP is a connectionless protocol and has no concept of state. Because of this, it is very easy to spoof source IP addresses in UDP as there is no initial setup that requires a response. This is different from a TCP connection where an initial three-way handshake is required and source spoofing is much more difficult.

However, it is unlikely that the source address was spoofed. There have been reports of some tools using UDP port 0 for OS fingerprinting (Stephens) or in a traceroute

(Rietveld), as will be shown is the case here. Because of this, the attacker will want to see the return traffic and will not spoof the source address.

4. Description of Attack:

There are a number of possibilities of what could be occurring here. First, this could be an active OS fingerprint scan. However, an OS fingerprint scan would most likely include other traffic to the host. Active OS fingerprinting programs work by sending a host a number of packets with unusual settings. Since every operating system responds to traffic in it's own way, the fingerprinting program can guess what operating system is responding to it by looking at how the return packets are formatted. However, as will be shown later, the only traffic between the two hosts are ICMP echo requests, DNS traffic and the UDP datagrams Snort alerted on; all of which do not seem to be unusual. Therefore, this probably isn't an OS fingerprinting scan.

A denial of service attack exists against some Checkpoint Firewall-1 machines where the firewall can crash if it receives a UDP packet destined for port 0. However, for this to occur, the attacker must be coming in through a VPN-1 interface. This did not occur here. Additionally, the destination host is not a Checkpoint Firewall-1 firewall.

The packets here are being used in a traceroute for high performance purposes. IP address 208.203.77.22 resolves to hostname mia-3dns.trialgraphix.com, which appears to be a 3-DNS machine. 3-DNS is a load-balancing solution created by F5 Networks where it "sends users to the best site based on rich performance metrics it collects from local area load balancers, servers and cache devices throughout the network." (F5 Networks) As shall be seen later, 3-DNS does some of its performance metrics using ICMP echo requests and the UDP datagrams that Snort picked up.

5. Attack mechanism:

If we look at all of the traffic coming from 208.203.77.22, we see how it executes its performance metrics and why Snort alerted on it. The initial traffic we see between 192.168.25.15 and 208.203.77.22 are some DNS requests and responses.

```
18:46:51.729255 192.168.25.15.1138 > 208.203.77.22.53: [udp sum ok]  23679
[1au] MX? trialgraphix.com. (45) (ttl 62, id 40178, len 73)
0x0000    4500 0049 9cf2 0000 3e11 796d c0a8 190f  E..I....>.ym.S..
0x0010    d0cb 4d16 0472 0035 0035 b465 5c7f 0010  ..M..r.5.5.e\...
0x0020    0001 0000 0000 0001 0c74 7269 616c 6772  .........trialgr
0x0030    6170 6869 7803 636f 6d00 000f 0001 0000  aphix.com.......
0x0040    2908 0000 0080 0000 00                   ).........

18:46:51.781581 208.203.77.22.53 > 192.168.25.15.1138: [udp sum ok]  23679
FormErr% [0q] 0/0/0 (12) (ttl 53, id 4783, len 40)
0x0000    4500 0028 12af 0000 3511 0cd2 d0cb 4d16  E..(....5.....M.
0x0010    c0a8 190f 0035 0472 0014 b7c9 5c7f 8091  .S...5.r....\...
0x0020    0000 0000 0000 0000 0000 0000 0000       ..............
```

```
18:46:51.782848 192.168.25.15.1138 > 208.203.77.22.53: [udp sum ok]  10485
MX? trialgraphix.com. (34) (ttl 62, id 40179, len 62)
0x0000    4500 003e 9cf3 0000 3e11 7977 c0a8 190f  E..>....>.yw.S..
0x0010    d0cb 4d16 0472 0035 002a 119f 28f5 0000  ..M..r.5.*..(...
0x0020    0001 0000 0000 0000 0c74 7269 616c 6772  .........trialgr
0x0030    6170 6869 7803 636f 6d00 000f 0001       aphix.com.....

18:46:51.833829 208.203.77.22.53 > 192.168.25.15.1138: [udp sum ok]
10485* 1/2/4 trialgraphix.com. MX smtp.trialgraphix.com. 10 (165) (ttl 53,
id 4784, len 193)
0x0000    4500 00c1 12b0 0000 3511 0c38 d0cb 4d16  E.......5..8..M.
0x0010    c0a8 190f 0035 0472 00ad 5a6f 28f5 8480  .S...5.r..Zo(...
0x0020    0001 0001 0002 0004 0c74 7269 616c 6772  .........trialgr
0x0030    6170 6869 7803 636f 6d00 000f 0001 c00c  aphix.com.......
0x0040    000f 0001 0001 5180 0009 000a 0473 6d74  ......Q......smt
0x0050    70c0 0cc0 0c00 0200 0100 0151 8000 0b08  p..........Q....
0x0060    6d69 612d 3364 6e73 c00c c00c 0002 0001  mia-3dns........
0x0070    0001 5180 000b 0863 6869 2d33 646e 73c0  ..Q....chi-3dns.
0x0080    0cc0 3000 0100 0100 0000 1e00 040c 2f01  ..0.........../.
0x0090    04c0 3000 0100 0100 0000 1e00 040c 2f01  ..0.........../.
0x00a0    04c0 4300 0100 0100 0151 8000 04d0 cb4d  ..C......Q.....M
0x00b0    16c0 5a00 0100 0100 0151 8000 040c 2f01  ..Z......Q..../.
0x00c0    03
```

The DNS requests and responses shown above are normal DNS traffic, initiated by 192.168.25.15 to 208.203.77.22. Since 192.168.25.15 is an SMTP server, most likely someone is sending an email to the trialgraphix.com domain. We know this because the DNS requests above are for the mail (MX) record for trialgraphix.com.

Two seconds after the DNS traffic stops, 208.203.77.22 initiates some ICMP echo requests to 192.168.25.15.

```
18:46:53.526091 208.203.77.22 > 192.168.25.15: icmp: echo request (ttl 53,
 id 4787, len 64)
0x0000    4500 0040 12b3 0000 3501 0cc6 d0cb 4d16  E..@....5.....M.
0x0010    c0a8 190f 0800 95c7 6238 0000 0000 0000  .S......b8......
0x0020    0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030    0000 0000 0000 0000 0000 0000 0000 0000  ................

18:46:54.535574 208.203.77.22 > 192.168.25.15: icmp: echo request (ttl 53,
 id 4820, len 64)
0x0000    4500 0040 12d4 0000 3501 0ca5 d0cb 4d16  E..@....5.....M.
0x0010    c0a8 190f 0800 93c7 6238 0100 0000 0000  .S......b8......
0x0020    0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030    0100 0000 0000 0000 0000 0000 0000 0000  ................

18:46:55.545815 208.203.77.22 > 192.168.25.15: icmp: echo request (ttl 53,
 id 4853, len 64)
0x0000    4500 0040 12f5 0000 3501 0c84 d0cb 4d16  E..@....5.....M.
0x0010    c0a8 190f 0800 91c7 6238 0200 0000 0000  .S......b8......
0x0020    0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030    0200 0000 0000 0000 0000 0000 0000 0000  ................
```

208.203.77.22 sends out three ICMP echo requests to 192.168.25.15. These are sent out to see the round trip time for traffic going from 208.203.77.22 to 192.168.25.15 and back. By measuring the round trip time, 208.203.77.22 can be sure to send 192.168.25.15 to a closer DNS or SMTP server, if one exists, the next time 192.168.25.15 submits a DNS request.

The ICMP packets themselves are normal, with the exception of bytes 0x1a and 0x30, highlighted above. These bytes are sequential counters for the ICMP packets, possibly for error checking or performance purposes.

Until this point, Snort has not alerted on anything since the traffic has been normal. However, after almost 30 seconds and no response to the ICMP echo requests, 208.203.77.22 sends out five UDP datagrams destined for port 0, which Snort alerts on.

```
18:47:23.860953 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36
 [ttl 1] (id 27417, len 64)
0x0000     4500 0040 6b19 0000 0111 e84f d0cb 4d16  E..@k......O..M.
0x0010     c0a8 190f 0035 0000 002c ad81 6b19 8081  .S...5...,..k...
0x0020     0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030     0000 0000 0000 0000 0000 0000 0000 0000  ................

18:47:24.866417 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36
 [ttl 1] (id 27418, len 64)
0x0000     4500 0040 6b1a 0000 0111 e84e d0cb 4d16  E..@k......N..M.
0x0010     c0a8 190f 0035 0000 002c ad80 6b1a 8081  .S...5...,..k...
0x0020     0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030     0000 0000 0000 0000 0000 0000 0000 0000  ................

18:47:25.876605 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36
 (ttl 2, id 27419, len 64)
0x0000     4500 0040 6b1b 0000 0211 e74d d0cb 4d16  E..@k......M..M.
0x0010     c0a8 190f 0035 0000 002c ad7f 6b1b 8081  .S...5...,..k...
0x0020     0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030     0000 0000 0000 0000 0000 0000 0000 0000  ................

18:47:26.886692 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36
 (ttl 2, id 27420, len 64)
0x0000     4500 0040 6b1c 0000 0211 e74c d0cb 4d16  E..@k......L..M.
0x0010     c0a8 190f 0035 0000 002c ad7e 6b1c 8081  .S...5...,.~k...
0x0020     0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030     0000 0000 0000 0000 0000 0000 0000 0000  ................

18:47:27.896698 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36
 (ttl 2, id 27421, len 64)
0x0000     4500 0040 6b1d 0000 0211 e74b d0cb 4d16  E..@k......K..M.
0x0010     c0a8 190f 0035 0000 002c ad7d 6b1d 8081  .S...5...,.}k...
0x0020     0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0030     0000 0000 0000 0000 0000 0000 0000 0000  ................
```

With the exception of the destination port of 0 and the small Time To Live (TTL) fields, the datagrams appear to be normal. The only payload in the packets is an increasing number, highlighted above, which correspond to the IP Identification number, and

followed by 0x8081. Like the ICMP echo requests packets that were first sent out, this sequencing is to keep track of the packets as they go out.

The purpose of the UDP datagrams is two-fold. First, the datagrams are being used to perform a traceroute to the destination machine. This can be seen in the increasing TTL fields of the packets. The first two datagrams have a TTL of 1, while the other three have a TTL of 2. When a packet's TTL reaches 0, the machine that the packet is at will send an ICMP Time To Live exceeded message to the source IP address instead of forwarding the packet on to the destination IP address. By starting off with a TTL of 1 and increasing it slowly, a machine can tell the path that it's traffic takes to get to the remote machine by examining the IP address of the sender of the ICMP TTL exceeded messages.

Here 208.203.77.22 is using the traceroute to tell the exact path it takes to get from itself to 192.168.25.15. This is useful if the reason the original ICMP echo requests failed was because a firewall was blocking the ICMP requests from getting to the destination in the first place. The 3-DNS server can examine the time it took for the last ICMP TTL exceeded message to get back to it and perform the evaluation on what the closest machine to use is based on that.

The datagrams are also being sent in the hopes that an ICMP Port Unreachable message will be sent back. ICMP Port Unreachable messages are sent whenever a UDP packet attempts to connect to a remote port that does not have a listening process attached to it. A Port Unreachable message would be generated here since the destination port is 0, a reserved port that nothing should ever listen on. If an ICMP Port Unreachable message is sent back from 192.168.25.15, 208.203.77.22 can execute the round trip time performance calculations using that.

To increase the likelihood an ICMP message is sent back, the UDP packets are formatted to bypass firewall restrictions by setting the source port to 53. Port 53 is used by DNS and many firewalls have been configured to allow UDP packets with source port 53 from any host into their network in order to allow DNS responses to get through. The 3-DNS server is hoping that by setting the source port to 53, any firewall will think this is a normal DNS response and let the packet through, thereby generating the ICMP port unreachable.

6. Correlations:

3-DNS queries like the ones Snort alerted on have been seen by a number of other people and have been reported on in Ronny Rietveld's GIAC GCIA Network Detect. In Ronny's Network Detect Analysis, he describes the same packets as were seen here. Ronny also concludes that this is due to a 3-DNS server.

The IP address 208.203.77.22 does not have any corresponding entries in the DShield database. This indicates that 208.203.77.22 either is not a malicious host or has not been seen by any host that reports to DShield yet.

7. Evidence of Active Targeting:

   The 3-DNS server, 208.203.77.22, is actively sending traffic to 192.168.25.15 to create performance metrics and is not doing a general scan or sending misdirected traffic.

8. Severity:

   The equation for severity is:

   severity = (criticality + lethality) – (system countermeasures + network countermeasures)

   Each value is rated on a scale from 1 to 5, where 1 is the lowest value.

   Criticality – The server being targeted, 192.168.25.15, is the primary Internet mail server. Since this machine is extremely critical to the infrastructure, criticality is given a value of 5.

   Lethality – Since the attack against the system would at most create an ICMP Port Unreachable error message, it is not considered lethal. At most, this would give an attacker an idea of what IP addresses were active. Lethality is given a value of 1.

   System Countermeasures – The mail server is a patched and protected system running a host-based IDS. Access to the machine is strictly regulated and its logs are closely monitored. Because of this, system countermeasures are given a value of 4.

   Network Countermeasures – The mail server is in a protected subnet guarded by a stateful firewall. The firewall is set up to only allow SMTP and DNS traffic to and from the mail server and nothing else. Because of this, the UDP datagrams and ICMP messages did not get through. Additionally, a border router is in place to do ingress and egress filtering on traffic coming from the Internet. Due to the strict protections in place, network countermeasures are given a value of 5.

   Severity = (5+1) – (4+5) = 6 – 9 = -3

   A severity score of –3 is a good score and indicates sufficient protections are in place.

9. Defensive Recommendation:

   To prevent traffic like this from reaching it's destination, setting up a stateful firewall to only allow necessary traffic through would be recommended. In this case the firewall is set up to only allow SMTP connections in and out of the machine and DNS traffic out. All other traffic is blocked. Setting up an Intrusion Detection System to monitor the traffic, as is done here, is also recommended.

   The defenses set up here are fine since the UDP datagrams sent to elicit an ICMP Port Unreachable error message were blocked. If the UDP datagrams had been able to get

through, the ICMP replies would have been blocked by the firewall as well due to the strict rule set in place.

10. Multiple Choice Test Question:

```
18:47:27.896698 208.203.77.22.53 > 192.168.25.15.0: [udp sum ok] udp 36
(ttl 28, id 27421, len 64)
 0x0000   4500 0040 6b1d 0000 1c11 e74b d0cb 4d16  E..@k......K..M.
 0x0010   c0a8 190f 0035 0000 002c ad7d 6b1d 8081  .S...5...,.}k...
 0x0020   0000 0000 0000 0000 0000 0000 0000 0000  ................
 0x0030   0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Given the packet above, what is the most likely response from the destination host?

A. TCP Reset
B. UDP DNS response
C. ICMP Port Unreachable error message
D. ICMP TTL Time Exceeded error message

Answer: C. According to RFC 1122, a host is supposed to send an ICMP Port Unreachable "when the designated transport protocol (e.g., UDP) is unable to demultiplex the datagram but has no protocol mechanism to inform the sender. "

11. References.

"F5 Networks – 3DNS Controller." F5 Networks. 11 Feb. 2003. <http://www.f5.com/f5products/3dns/>.

"Firewall-1 Port 0 Denial of Service Vulnerability." SecurityFocus Vulnerability Database. 11 Feb. 2003. <http://online.securityfocus.com/bid/576/info/>.

"HyperRFC: file rfc1122.txt." 11 Feb 2003. <http://www.csl.sony.co.jp/cgi-bin/hyperrfc?1122>.

Rietveld, Ronny. "LOGS: GIAC GCIA Version 3.3 Practical Detect#3 (Rietveld)." 27 Oct. 2002. Intrusions mailing list. 11 Feb 2003. <http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00318.html>.

Stephens, Alex. Alex Stephens (LevelTwo SANS GCIA Certification Exam. 25 Mar. 2001. Global Information Assurance Certification. 11 Feb. 2003. <http://www.giac.org/practical/Alex_Stephens_GCIA.htm>.

Detect #3 – Kuang2 virus SYN Scans

On the night of February 12, 2003 starting at 20:42:12 and proceeding until 03:28:51 the morning of February 13, 2003, my employer's network was scanned 1,999 times by 33 different IP addresses looking for any machine listening on port 17300. This port is associated with Kuang2 theVirus.

The following are some of the packets detected by the Shadow IDS:

```
20:24:12.881873 66.138.126.219.4764 > 192.168.25.2.17300: S
1977475:1977475(0) win 8192 (DF)

20:24:12.913925 66.138.126.219.4766 > 192.168.25.4.17300: S
1977508:1977508(0) win 8192 (DF)

20:39:07.997189 220.89.88.232.2562 > 192.168.25.24.17300: S
1724452:1724452(0) win 8192 (DF)

20:39:08.150916 220.89.88.232.2544 > 192.168.25.6.17300: S 1721632:1721632(0)
win 8192 (DF)

02:26:03.289354 211.224.18.12.3376 > 192.168.25.5.17300: S 4692294:4692294(0)
win 34930

02:26:03.612249 211.224.18.12.3398 > 192.168.25.27.17300: S
4695601:4695601(0) win 34930
```

The following are selected full packet dumps from the Shadow tcpdump logs, obtained using tcpdump:

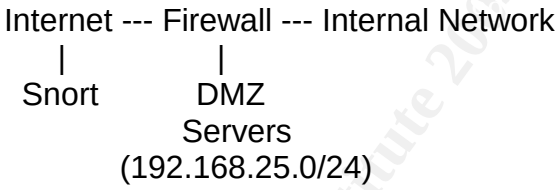
```
$ tcpdump –r log –n –s 1514 -X
20:24:12.881873 66.138.126.219.4764 > 192.168.25.2.17300: S
1977475:1977475(0) win 8192 <mss 1414,nop,nop,sackOK> (DF)
0x0000   4500 0030 9f29 4000 7106 60e3 428a 7edb      E..0.)@.q.`.B.~.
0x0010   c0a8 1902 129c 4394 001e 2c83 0000 0000      ......C...,.....
0x0020   7002 2000 d6c0 0000 0204 0586 0101 0402      p...............

20:24:12.913925 66.138.126.219.4766 > 192.168.25.4.17300: S
1977508:1977508(0) win 8192 <mss 1414,nop,nop,sackOK> (DF)
0x0000   4500 0030 a329 4000 7106 5ce1 428a 7edb      E..0.)@.q.\.B.~.
0x0010   c0a8 1904 129e 4394 001e 2ca4 0000 0000      ......C...,.....
0x0020   7002 2000 d69b 0000 0204 0586 0101 0402      p...............

20:39:07.997189 220.89.88.232.2562 > 192.168.25.24.17300: S
1724452:1724452(0) win 8192 <mss 1460,nop,nop,sackOK> (DF)
0x0000   4500 0030 4d42 4000 6f06 40d8 dc59 58e8      E..0MB@.o.@..YX.
0x0010   c0a8 1918 0a02 4394 001a 5024 0000 0000      ......C...P$....
0x0020   7002 2000 479d 0000 0204 05b4 0101 0402      p...G...........

20:39:08.150916 220.89.88.232.2544 > 192.168.25.6.17300: S 1721632:1721632(0)
win 8192 <mss 1460,nop,nop,sackOK> (DF)
0x0000   4500 0030 4f42 4000 6f06 3eea dc59 58e8      E..0OB@.o.>..YX.
0x0010   c0a8 1906 09f0 4394 001a 4520 0000 0000      ......C...E.....
```

```
0x0020   7002 2000 52c5 0000 0204 05b4 0101 0402      p...R..........

02:26:03.289354 211.224.18.12.3376 > 192.168.25.5.17300: S 4692294:4692294(0)
win 34930 <mss 1414,nop,wscale 3,nop,nop,timestamp 0 0,nop,nop,sackOK>
0x0000   4500 0040 9171 0000 3106 ca01 d3e0 120c      E..@.q..1.......
0x0010   c0a8 1905 0d30 4394 0047 9946 0000 0000      .....0C..G.F....
0x0020   b002 8872 9522 0000 0204 0586 0103 0303      ...r."..........
0x0030   0101 080a 0000 0000 0000 0000 0101 0402      ................

02:26:03.612249 211.224.18.12.3398 > 192.168.25.17300: S 4695601:4695601(0)
win 34930 <mss 1414,nop,wscale 3,nop,nop,timestamp 0 0,nop,nop,sackOK>
0x0000   4500 0040 9471 0000 3106 c6eb d3e0 120c      E..@.q..1.......
0x0010   c0a8 191b 0d46 4394 0047 a631 0000 0000      .....FC..G.1....
0x0020   b002 8872 880b 0000 0204 0586 0103 0303      ...r............
0x0030   0101 080a 0000 0000 0000 0000 0101 0402      ................
```

1. Source of Trace:

This detect was obtained from a Shadow IDS listening to my employers Internet connection.

Without going too deeply into the structure of the network, a Shadow sensor running tcpdump logs all of the Internet traffic outside of the firewall through an interface in stealth mode. This means the IDS sensor's interface has no IP address assigned to it so it can only listen to traffic and not respond. Behind the firewall resides the protected e-commerce servers and internal network. The following is a basic diagram of how the network is organized.



It should be noted that in this alert the destination IP address, 192.168.25.15, is obfuscated to hide the actual class C Internet IP address and internal IP address of the machine in the alert. The source IP addresses have not been changed.

2. Detect was generated by:

The detect was generated by a Shadow IDS sensor. Shadow is maintained by the Naval Surface Warfare Center (NSWC) and is located at http://www.nswc.navy.mil/ISSEC/CID/index.html.

Shadow works by having a sensor log all network traffic it sees using tcpdump. Once an hour, the sensor will securely copy the logs using OpenSSH to another machine where Shadow scripts will run the tcpdump logs through pre-defined BPF filters created by a Shadow IDS analyst. The filters are set up to remove known good traffic, such as SMTP traffic to a mail server or HTTP traffic to a web server. Any traffic that is left is

not normal and therefore suspect. The filters are also set up to find any unusual network traffic, such as TCP packets with the SYN and FIN flags set at the same time.

After Shadow runs the tcpdump logs through the filters, the resulting traffic is added to a web page the Shadow analyst can view. The format of the resulting traffic, as shown below, is what one would see if they were looking at the traffic through tcpdump. Since the original tcpdump logs are kept, the Shadow analyst can go back and analyze any traffic further.

The following are more of the port 17300 scans taken directly from the Shadow HTML page.

```
12.207.21.138 > 192.168.25.2

20:59:50.183084 12.207.21.138.4991 > 192.168.25.2.17300: S 3414363:3414363(0)
win 65535 (DF)

20:59:50.896227 12.207.21.138.4994 > 192.168.25.5.17300: S 3415075:3415075(0)
win 65535 (DF)

20:59:50.900041 12.207.21.138.4995 > 192.168.25.6.17300: S 3415078:3415078(0)
win 65535 (DF)

20:59:51.559479 12.207.21.138.4999 > 2.168.25.10.17300: S 3415738:3415738(0)
win 65535 (DF)

20:59:51.562770 12.207.21.138.5000 > 192.168.25.11.17300: S
3415741:3415741(0) win 65535 (DF)
```

3. Probability the source address was spoofed:

I do not believe the source addresses of any of the packets were spoofed for a number of reasons. First, each packet that is looking for TCP port 17300 has only the SYN flag set. This means that the attacker will be expecting a SYN/ACK packet returned for an open port or a RST packet for a closed port, as per RFC 793. If the attacker were spoofing their IP address, they would not see this return traffic and would not know if the port was open.

While there is the possibility that some of the source addresses are spoofed to act as decoys and hide the true IP address of the scanner, I do not think that this is the case here. First of all, each of the 33 different source IP addresses scan for port 17300 at different times, with no packets from one source address starting before a different source address has completely finished. If an attacker were spoofing some of the source IP addresses as decoys, they would send the decoy packets along with the real packets at the same time.

Additionally, every source IP address does not scan every destination IP address. If some of the probes were decoy packets, they would all be scanning the same IP addresses.

Finally, many of the fields that are typically unusual with spoofed packets are not. Since the packets shown below are typical of all the packets seen in the scan, they will be examined.

In the packets below, the sequence number in the TCP header varies with each source address, indicating the packets are coming from different machines. With tools such as nmap, the sequence number is the same in the spoofed packet as the good packet. The Time To Live (TTL) field in each packet, highlighted as byte 0x08 below, is also normal. In spoofed packets, the TTL will occasionally be unusually low or high. In the packets below, the TTL's are 112 and 113 respectively, a reasonable TTL for a packet starting with a TTL of 128.

Finally, the source ports below are not below 1024, which can indicate a spoofed packet. This is because machines will not normally create packets with a source port less than 1024, since these ports are reserved. There are, of course, a few exceptions to this.

```
23:21:59.829997 218.52.154.221.2451 > 192.168.25.58.17300: S 308387:308387(0)
win 8192 <mss 1460,nop,nop,sackOK> (DF)
0x0000   4500 0030 700a 4000 7006 dd1d da34 9add     E..0p.@.p....4..
0x0010   c0a8 193a 0993 4394 0004 b4a3 0000 0000     ...:..C.........
0x0020   7002 2000 a3b0 0000 0204 05b4 0101 0402     p...............

00:01:59.270945 211.51.125.249.1552 > 192.168.25.31.17300: S
1693664:1693664(0) win 8192 <mss 1460,nop,nop,sackOK> (DF)
0x0000   4500 0030 bf31 4000 7106 b0f6 d333 7df9     E..0.1@.q....3}.
0x0010   c0a8 191f 0610 4394 0019 d7e0 0000 0000     ......C.........
0x0020   7002 2000 a7e1 0000 0204 05b4 0101 0402     p...............
```

4. Description of attack:

This is a SYN scan looking for computers infected with Kuang2 theVirus, which listens on TCP port 17300. Kuang2 is a trojan horse for Windows that allows a remote attacker to upload, download, delete and run files on an infected machine. The trojan horse also has the ability to load plug-ins to provide more functionality. Additionally, tools like Kuang2 Web Updater exist that will scan a list of IP addresses for Kuang2 infected hosts and force them to download and run a program. This makes a mass-compromise of infected hosts quick and easy to perform.

Since the scan is coming from many different IP addresses during a relatively short amount of time, the scan is probably controlled by one attacker and distributed across all of the machines detected in the scan. This makes sense since Kuang2 provides the ability to remotely upload and run programs on infected machines. An attacker who had control of a number of infected machines could use Kuang2 to upload a scanning program, such as Kuang2 Web Updater, and scan for other infected machines.

Running p0f against the tcpdump logs strengthens this theory. P0f is a passive OS fingerprinting tool that will listen to a network interface or examine a tcpdump log and try

to determine what operating system the sender of every TCP SYN packet is running. Every operating system has it's own unique way of creating a TCP connection and p0f looks for these nuances within every SYN packet to determine, if possible, the operating system.

When run against the SYN scan, p0f determined that 24 out of the 33 hosts were running a Windows operating system; the operating system that Kuang2 infects. This means that the hosts that were scanning for Kuang2 could have been infected with Kuang2 and used as described above. P0f was not able to determine the operating system of the other 9 hosts.

Of course, the scans could be coming from different attackers. However, it is very unusual that the scans never overlap and abruptly stop. If these were indeed random scans from different attackers, one would expect the scans to trickle on over time and not within a specific time space. Additionally, since these scans occurred, no other scans for port 17300 have been seen. This implies that one person or group was controlling all of the scans.

5. Attack mechanism:

A SYN scan occurs when an attacker sends out TCP packets with only the SYN flag set for a specific port to a number of remote machines. The attacker is relying upon the way TCP works to find out if the remote port is available.

In TCP connections, a three-way handshake is required to start a conversation. The first TCP packet sent has only the SYN flag set, signifying that the client would like to talk to the remote server on the destination service. If the remote service is available, the server will send back a TCP packet with the SYN and ACK flags set. The client will then respond with a third packet with just the ACK flag set.

If the remote service is not available, instead of sending a SYN/ACK packet the server will send a TCP packet with the RST flag set. The purpose of the three-way handshake is to synchronize the sequence numbers on both machines so a reliable connection can be provided.

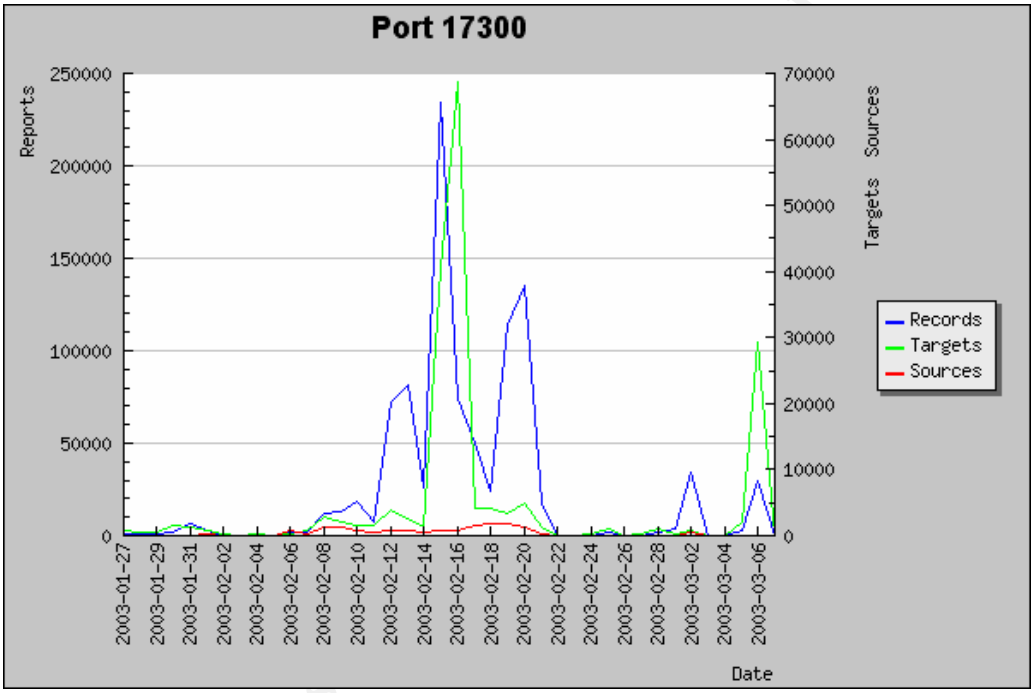
The attacker is sending out TCP SYN packets for port 17300 to the remote IP addresses, hoping that a SYN-ACK or RST response will be sent back. In doing so the attacker will quickly be able to tell if a remote machine is infected with Kuang2.

This scan is probably not done with a port scanning tool such as nmap but with a specialized tool like the Kuang2 Web Updater, which will scan a list of infected IP addresses and force any infected machine to download a file from a URL and run it. Using a tool such as this would allow an attacker to quickly take over a large number of hosts.

6. Correlations:

TCP SYN scans are common, but during the month of February, scans for port 17300 increased and were detected by many analysts. For example, on February 13, Chris Jones posted to the Intrusions mailing list that he had seen many port 17300 scans directed against his network the previous night.

As shown below, the incidents.org port report for 17300, which feeds off of the statistics provided by DShield, shows a massive jump in scanning for port 17300 beginning around February 6, 2003 with the peak occurring on February 16 with close to 70,000 targets. During the entire jump in scanning, the number of sources never exceeded 1925 hosts. This again suggests a coordinated scan was taking place.



(Image used with permission from incidents.org)

The DShield database did not have any entries for any of the 33 IP addresses. This implies that these machines have not been used for scanning before, or no one had reported them as scanning.

Searching the Internet revealed that Glenn Larratt posted on the Intrusions mailing list that he received what he believes to be a coordinated scan for TCP 17300 in July 2002. All of the scans Larratt saw occurred over a short period of time like the scan detected here.

7. Evidence of active targeting:

The scan detected is a general scan of a class C network looking for machines infected with the Kuang2 virus. It is directed to this class C network, but in all probability, it was scanning IP network ranges around it as well.

8. Severity:

The equation for severity is:

severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Each value is rated on a scale from 1 to 5, where 1 is the lowest value.

Criticality – The network being scanned was the entire class C network of the company. Every machine on this network is extremely critical to the success of the business. Therefore, criticality is assigned a value of 5.

Lethality – Since this was just a scan, a successful attack would have only resulted in the knowledge that an infected machine was present. This gives lethality a low score of 1.

System countermeasures – All hosts on the network that was scanned are regularly updated and monitored constantly. There is also host based intrusion detection software residing on each machine. Additionally, none of the systems on this network are running Microsoft Windows and cannot be infected by Kuang2. Therefore, system countermeasures are given a score of 5.

Network countermeasures – Each host is behind a stateful firewall and monitored via multiple network intrusion detection sensors. Since TCP port 17300 is not a service offered by any server, it is blocked at the firewall. Additionally, a border router is in place to do ingress and egress filtering on traffic coming from the Internet. Due to the strict protections in place, network countermeasures is given a value of 5.

Severity = (5+1) – (5+5) = 6 – 10 = -4

A severity score of –4 is a good score and indicates that sufficient protections are in place.

9. Defensive recommendations:

Preventing scans like the ones detected above requires using a firewall set to deny all connections except those for services offered or required. Additionally, using an intrusion detection system to detect the scans is also recommended. Since the scans are looking for a port that is used by a trojan horse, any Windows servers that are Internet accessible should be patched, have updated anti-virus software and host-based IDS installed.

Given that this scan was blocked by a stateful firewall and sufficient intrusion detection is in place and detected the scan, there are no defensive recommendations.

10. Multiple choice test question:

Which of the following packets from a tcpdump log file would the passive OS fingerprinting tool p0f look at to determine an operating system?

```
A. 11:59:52.445407 10.215.26.200 > 66.218.71.63: icmp: echo request

B. 11:00:57.608203 192.169.25.30.3837 > 208.45.133.230.53:  60466 [1au] A?
   test.myhost.com. (43)

C. 11:03:47.752865 207.54.158.16.34703 > 172.16.135.15.113: . ack 37 win
   24820 (DF)

D. 11:00:57.668668 12.111.239.35.8648 > 10.104.3.20.80: S
   3301816628:3301816628(0) win 512 <mss 1460>
```

Answer: D. P0f determines the operating system of a machine by looking at different settings in TCP SYN packets.

11. References:

Aphex. "Kuang2 Web Updater 1.1." Aug 2002. 2 April 2003. <http://www.megasecurity.org/trojans/k/kuang2_webupdater/Kuang2_webupdater1.1.html>.

"Internet Storm Center Port 17300 Report" 2 April 2003 <http://isc.incidents.org/port_details.html?port=17300&recax=1&tarax=2&srcax=2&percent=N&days=40>.

Jones, Chris. "port 17300 scanning?" 13 Feb 2003. Intrusions mailing list. 2 April 2003 <http://cert.uni-stuttgart.de/archive/intrusions/2003/02/msg00120.html>.

"Kuang2 the Virus 0.21." 2 April 2003 <http://www.dark-e.com/archive/trojans/kuang/tv/index.shtml>.

Larratt, Glenn Forbes Fleming. "[LOGS] new port 17300 scans." 29 July 2002. Intrusions mailing list. 2 April 2003 <http://cert.uni-stuttgart.de/archive/intrusions/2002/07/msg00226.html>.

"RFC 793: Transmission Control Protocol." Sept 1981. Jon Postel ed. 2 Apr 2003. <http://www.ietf.org/rfc/rfc0793.txt?number=793>.

"Security Port Scanner, Trojan Port List: Kuang2 the virus." 2 April 2003. <http://www.glocksoft.com/trojan_list/Kuang2_the_virus.htm>

Tharakan, Royans. "port 17300 probe fingerprint analysis." 17 Feb 2003. Intrusions mailing list. 2 April 2003 <http://cert.uni-stuttgart.de/archive/intrusions/2003/02/msg00172.html>.

Analyze This!

Executive Summary

An analysis was done on the university's Intrusion Detection System (IDS) logs to discover what security problems, if any, are present. This paper presents the findings of the analysis.

While examining the university's logs, a number of problems became apparent. First, peer to peer file-sharing applications, such as KaZaa and Gnutella, generate a large portion of the network traffic. These programs allow users to exchange files with each other on the Internet easily.

More often than not, however, the file sharing networks are utilized to download copyrighted MP3 music files and pirated software, which may present legal problems to the university. Allowing these programs to continue also opens the possibility for widespread virus infections to occur, as viruses and other malware are rampant on these networks. Additionally, this traffic causes many false positives to be generated by the IDS, which causes IDS analysts to waste time researching what is actually non-malicious traffic.

A number of internal hosts have also been compromised and are being used in Distributed Denial of Service (DDoS) attacks against external machines. This not only causes problems for the machine being attacked, but it also consumes bandwidth on the university network, triggering network slowness and other problems.

Logs Analyzed

The following logs were provided by the university to analyze:

alert.030215.gz	scans.030215.gz	OOS_Report_2003_02_16_32309
alert.030216.gz	scans.030215.gz	OOS_Report_2003_02_17_6137
alert.030217.gz	scans.030215.gz	OOS_Report_2003_02_18_27913
alert.030218.gz	scans.030215.gz	OOS_Report_2003_02_19_479
alert.030219.gz	scans.030215.gz	OOS_Report_2003_02_20_28598

The alert logs contain alerts generated by Snort with an unknown rule set. I believe that Snort version 1.7 or earlier was used, due to the wording of some of the different preprocessor alert messages. In the analysis of the alert files in the rest of this document, any messages generated by the spp_portscan Snort pre-processor contained in the alert logs are ignored since these alerts are also contained in the scan logs

The scan logs contain a listing of all of the portscans detected and were created by Snort's spp_portscan preprocessor.

The out-of-spec (OOS) logs contain TCP packets that have unusual flags set within them. The format of these files is the same as seen when 'snort –dv' is run. It should be noted that the dates of the OOS files examined are one off from the other files examined. This is because the data in the OOS files are one day off from the filename. For example, the data is OOS_Report_2003_02_20_28598 is actually from the 19th, not the 20th.

Alert logs analysis

A total of 146,499 alerts were present in the alert log files. The following table lists the alerts generated, ordered by total occurrence. The number of occurrences of internal and external sources for each alert is also broken out.

Total	Internal Src	External Src	Alert Name
74876		74876	SMB Name Wildcard
15079	14814	265	Incomplete Packet Fragments Discarded
12630		12630	Watchlist 000220 IL-ISDNNET-990517
6558		6558	CS WEBSERVER - external web traffic
5892	5446	446	spp_http_decode: IIS Unicode attack detected
5838	3716	2122	High port 65535 tcp - possible Red Worm - traffic
5782		5782	SUNRPC highport access!
3481	3457	24	spp_http_decode: CGI Null Byte attack detected
2737		2737	TCP SRC and DST outside network
1947	878	1069	TFTP - Internal TCP connection to external tftp server
1644		1644	Null scan!
1493	1493		TFTP - External UDP connection to internal tftp server
1360		1360	Watchlist 000222 NET-NCFC
971	491	480	High port 65535 udp - possible Red Worm - traffic
792	594	198	Port 55850 tcp - Possible myserver activity - ref. 010313-1
760		760	MY.NET.30.4 activity
629		629	Queso fingerprint
536		536	IDS552/web-iis_IIS ISAPI Overflow ida nosize
492	25	467	Tiny Fragments - Possible Hostile Activity
447	266	181	Possible trojan server activity
433		433	connect to 515 from outside
335		335	EXPLOIT x86 NOOP
283		283	TCP SMTP Source Port traffic
281		281	NETBIOS NT NULL session
214		214	External RPC call
178		178	MY.NET.30.3 activity
175		175	CS WEBSERVER - external ftp traffic
148	148		IRC evil - running XDCC
89	33	56	TFTP - External TCP connection to internal tftp server
89		89	NMAP TCP ping!
78		78	EXPLOIT x86 setuid 0
53		53	EXPLOIT x86 stealth noop
36	36		IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize
28		28	EXPLOIT x86 setgid 0
26		26	SNMP public access
26		26	Notify Brian B. 3.54 tcp
17	14	3	TFTP - Internal UDP connection to external tftp server
17		17	Notify Brian B. 3.56 tcp
17		17	Attempted Sun RPC high port access
6		6	Probable NMAP fingerprint attempt
6	6		Port 55850 udp - Possible myserver activity - ref. 010313-1
5		5	FTP passwd attempt
4		4	SMB C access
2	1	1	RFB - Possible WinVNC - 010708-1
2		2	PHF attempt
1		1	Fragmentation Overflow Attack

Top 10 internal alert talkers

# alerts	IP address
13193	MY.NET.211.6
1018	MY.NET.132.42
778	MY.NET.207.214
735	MY.NET.204.74
589	MY.NET.201.146
550	MY.NET.202.226
469	MY.NET.226.22
452	MY.NET.237.238
439	MY.NET.212.22
424	MY.NET.242.250

Top 10 external alert talkers

# alerts	IP address
4726	169.232.84.146
2187	212.179.123.163
1779	12.35.158.199
1281	212.179.88.96
1156	212.179.105.210
967	212.179.91.129
809	141.157.254.236
763	159.226.5.220
682	66.72.199.111
604	141.156.242.139

Top 5 alerts from external sources

The alerts analyzed below are the top five alerts generated by external sources. By analyzing the most numerous alerts from external sources, one can see what is being attacked and / or exploited the most on the internal network and base defensive recommendations on that. Additionally, studying the top alerts from external sources can give hints as to what security trends are occurring in the world. For example, if a large number of "SMB Name Wildcard" alerts are being generated, it could indicate something in the wild, such as a new virus or worm, is attempting to find and attack Microsoft Windows shares.

SMB Name Wildcard
Total alerts reported: 74,876
External sources: 74,876 Unique external sources: 16,365
Internal sources: 0 Unique internal sources: 0

The "SMB Name Wildcard" alert occurs when an attempt to list a remote Windows machine's NetBIOS name table is seen, such as when the Windows command "nbtstat –A <IP Address>" is used. In doing this, a lot of information about the machine can be revealed, including the machine's NetBIOS name and workgroup, user ID of any users logged in, and if any shares are open. All of this information can lead to a compromise of an unsecured machine.

This signature is seen a lot now since many attackers and viruses attempt to find any open Windows file shares on computers to attack. CERT issued an incident note on March 3, 2000 entitled "Exploitation of Unprotected Windows Networking Shares" to this effect. Since this alert was set off by so many different external sources, it leads one to believe that the university is not under a coordinated attack, but is feeling the brunt of the numerous worms in the wild that attempt to do this.

It is interesting that this alert only reported on external sources. This probably means that the IDS was configured to ignore internal machines setting off this alert. Since this alert can be generated due to normal traffic on an internal network, it is a good idea to

set your IDS to ignore internal SMB wildcard alerts; else you may be flooded with false positives.

Correlations: Tod Beardsley noted this event in his GIAC practical, although he saw mostly alerts generated from internal machines. This indicates one of two things about the current network: either the current network is allowing NetBIOS traffic from the internet onto it's internal network or the IDS is in a position where it is looking at the Internet traffic before it gets filtered by a firewall. With little knowledge of the internal network's structure, we could only guess at this answer.

Bryce Alexander also describes this alert in the SANS Port 137 Intrusion Detection FAQ, although he describes it in relation to the "network.vbs" worm.

Recommendations: It is recommended that a firewall be configured to block all NetBIOS traffic coming in from the Internet, if this is not done already. Since this alert is frequently set off due to worms and attackers scanning for open shares on Windows machines, it may be a good idea to turn off this alert once firewall blocking is set up. By leaving this rule on, a lot of alerts will be generated which will have no bearing on the security of the network.

Watchlist 000220 IL-ISDNNET-990517

Total alerts reported: 12,630	
External sources: 12,630	Unique external sources: 176
Internal sources: 0	Unique internal sources: 0

The "Watchlist 000220 IL-ISDNNET-990517" alert was created by the university and is not an alert provided by Snort. While we don't have access to the actual signature that triggers this alert, the signature refers to the whois handle "IL-ISDNNET-990517", which is the handle for ISDNet LTD, an Internet service provider in Israel. This signature probably exists because suspicious activity has been seen from this network before.

Looking at the alerts we see that there are only 176 unique sources out of 12,630 alerts. However, 4 of these sources are in the top 10 alert talkers and account for 45% of the "Watchlist 000220 IL-ISDNNET-990517" alerts. Two of these sources have the majority of their traffic going to ports 1214 and 4662, ports used by the peer to peer file sharing programs KaZaa and eDonkey, respectively. The other machines did not have any recognizable ports that they were sending data to, but since many file-sharing programs are able to listen on any port, there is a good chance that these were also part of a file-sharing network.

Additionally, port 80 (HTTP) is the most common source port within the alerts. This indicates that there are many web servers within this IP range that users on the university's network are going to.

Correlations: Brian Coyle saw a lot of traffic generating this alert in his GCIA analysis and concluded that much of it was due to file sharing and web traffic. Mike Worman

also reported seeing Napster traffic from this network in his analysis. This indicates that a lot of users that are using file-sharing programs, such as KaZaa or Napster, are contacting machines in the IP range above.

At the time of this writing, DShield's database does not show any malicious activity coming from the 212.179.0.0/16 network.

Recommendations: Since most of the traffic coming from ISDNet LTD is peer to peer file sharing or web based, there are only a few defensive recommendations that can be made. First, blocking all traffic coming from or going to 212.179.0.0/16 at the firewall will effectively stop any threat from this network, including the web and file sharing traffic currently occurring.

Eliminating the use of peer to peer file sharing, which the majority of the alerts appear to be generated from, is very difficult to accomplish. Using a combination of policy and end user education is somewhat effective. Joining this with blocking ports at the firewall that peer to peer programs communicate on will also help stop much of the traffic to and from this network.

CS WEBSERVER - external web traffic

Total alerts: 6,558

External sources: 6,558	Unique external sources: 2,885
Internal sources: 0	Unique internal sources: 0

This is another custom alert developed by the university. The alert was generated 6,558 times but only has 1 destination, MY.NET.100.165. All of the alerts were to port 80, HTTP.

The alert has been written to signal whenever an external IP address accesses the IP address specified above on port 80. This fact that this will only go off when traffic is sent to port 80 can further be seen as there are additional alerts in the logs for external traffic going to MY.NET.100.165 on ports other than 80.

The CS web server is most likely a web server owned by the computer science (CS) department and is either a frequent target of attacks or contains highly sensitive or important information. Since computer science students are more likely to fool around and hack, the computer science web server would be a prime target to attack and needs to be monitored.

The top external source that set this alert off, 141.157.254.236, resolves to pool-141-157-254-236.ny325.east.verizon.net. This is in an IP address space owned by Verizon Internet Services and is probably a dial-up or broadband Internet connection. The CS web server, MY.NET.100.165, was accessed by this address 809 times over a period of two days (approximately 7 hours total). This could be a student or faculty member accessing the server, or it could be a probe into the weaknesses present on the server.

Correlations: Scott Baird noted in his analysis that 18,080 alerts were generated with this signature. He also notes that there was only one destination for this alert, MY.NET.100.165, and that the signature was probably written specifically to notify when external traffic to that server was generated.

There were no other alerts from 141.157.254.236 in the university's logs and DShield's database did not note any malicious activity from it. Searching Google for the IP address did not generate anything either. This could indicate that whoever was using this address was harmless, or had not been caught yet.

Recommendations: If the CS web server is a prime target of attack from external sources, it is recommended to block all access to the server from any external traffic with a stateful firewall. Additionally, keeping the server patched and installing security software, like anti-virus or intrusion detection, onto it will help prevent any compromises. Reviewing the web logs of the server will also help catch anyone attempting to compromise the server.

SUNRPC highport access!

Total alerts reported: 5,782

External sources: 5,782	Unique external sources: 25
Internal sources: 0	Unique internal sources: 0

This alert is generated whenever traffic to port 32771 is seen. Looking through the logs at this alert, there are 25 unique, external sources, going to port 32771 on 14 different internal destinations. ISS notes that on Sun servers, rpcbind will listen on port 32771 in addition to port 111. This is a problem as an attacker could obtain SUNRPC program information even if the standard port for rpcbind, port 111, is filtered.

However, port 32771 is above 1024, and could be used by a client in a normal client / server connection. Therefore, some of the alerts generated could be false positives. Scanning through the source ports from the 25 source addresses, there are a number of ports that are associated with normal services. Specifically, ports 20 (passive FTP), 22 (SSH), 80 (HTTP), 1214 (KaZaa), 5190 (AOL Instant Messenger) and 6667 (X11) are present. The alerts associated with these connections are false positives and not malicious.

After filtering out the false positives, we are left with 5,467 alerts coming from 5 unique sources. Out of the remaining sources, two of them, 169.232.84.146 and 66.72.199.111, appear on the top alert talkers list with 4726 and 682 alerts, respectively. Both of these machines talk exclusively to one internal source, MY.NET.252.126.

Each of these machines generates alerts to MY.NET.252.126 for approximately four hours total. Since so much traffic is going back and forth between these hosts, it could indicate something malicious is occurring, especially if this machine is not running SUNRPC services.

Correlations: Many GCIA analyst reports mention seeing the "SUNRPC highport access!" alert in many alert files. As seen here, James Hoover reports in his analysis seeing this alert generated as a false positive. In his case, this alert was set off from a normal telnet session. However, James also reports on seeing this alert set off due to a host that has probably been compromised.

The top two talkers for this alert, 169.232.84.146 and 66.72.199.111, did not have any other alerts and had no corresponding entries in the DShield database.

Recommendations: Since it is possible that a Sun server could be running rpcbind on port 32771, a stateful firewall should be set up to block external traffic to internal machines. Additionally, any Sun servers affected by this should turn off rpcbind if it is not needed or apply the patches supplied by Sun.

It also appears that IP address MY.NET.252.126 may have been compromised. This machine should be taken off-line and forensically analyzed to see if any compromise has occurred.

TCP SRC and DST outside network

Total alerts reported: 2,737

External sources: 2,737	Unique external sources: 2,247
Internal sources: 0	Unique internal sources: 0

This is another custom alert and although we don't have access to the rule that generates this alert, we can speculate that it is generated when a TCP packet with a source and destination IP address outside of the internal network is seen. Looking at the IP addresses in the alerts, there are no IP addresses for the "MY.NET" network, so this conclusion is the correct one.

Seeing traffic with the IP source and destination set to outside of the current network indicates that external traffic is entering the internal network or IP spoofing is occurring. IP spoofing takes place when an attacker changes, or spoofs, their source IP address to make it look like the network traffic they are sending is coming from a different machine. IP spoofing is most commonly used in denial of service attacks or as a technique for hiding one's true IP address amongst garbage traffic, such as with tools like Nmap. IP spoofing can also be used for exploits, but this is very difficult to do with TCP traffic and is usually not seen.

In all of the alerts seen, there are only 58 unique destination hosts. The top destination that traffic is sent to is 216.209.164.171, with 1,790 alerts. This host resolves to newmarket-ppp277234.sympatico.ca and appears to be a Canadian DSL account. Every packet sent to this host is destined for TCP port 135, an end point mapper used on Microsoft Windows machines. Additionally, there are 1,790 different, spoofed source addresses, but every source address is in the 171.165.0.0/16 subnet that is owned by Bank of America.

What is happening is an attacker is trying to denial of service 216.209.164.171, making it look like Bank of America is doing the attack. This could mean that an internal host or hosts are compromised and are being used in the attack or an internal user is attacking 216.209.164.171.

Correlations: Many GCIA analysts have indicated seeing this alert in their analyses of the university's traffic. Michael McDonnell wrote in his analysis about seeing this alert, but the majority of his traffic was to non-routable IP addresses.

The DShield database shows no reports of malicious activity for the 171.165 subnet.

Recommendations: This alert indicates IP spoofing is taking place or external traffic is getting into the internal network. To prevent this type of traffic from entering or leaving the internal network, ingress and egress filtering should be set up any border routers leading to the internal network from the Internet or any 3rd party connection. This will prevent traffic destined for non-internal IP addresses from entering the network and traffic coming from non-internal IP addresses from exiting the network.

Additionally, IP spoofing can indicate a compromised machine. Tracing back a spoofed source to it's true IP address is done by following MAC addresses in conjunction with the traffic, and is nearly impossible to do after the traffic has stopped. If this traffic starts again, a procedure should be set up to trace this traffic back to find out who, and why, it is being generated.

Top 5 alerts from internal sources

The alerts analyzed below are the top 5 alerts where the majority of the source IP addresses are from the internal network. Alerts generated from internal machines indicate where security compromises have occurred or where network problems lie. While the alerts analyzed below might have additional alerts generated from external sources, these will not be looked at and instead the problems associated with internal sources generating these alerts will be focused on.

Incomplete Packet Fragments Discarded

Total alerts reported: 15,079

External sources: 265	Unique external sources: 44
Internal sources: 14,814	Unique internal sources: 6

The "Incomplete Packet Fragments Discarded" alert occurs when Snort's spp_defrag preprocessor detects a packet that has been fragmented, but is not able to reassemble the entire packet due to missing fragments. Snort discards the rest of the fragments because without the full packet, the IDS cannot analyze it. Missing fragments only occur in a few situations.

The first reason an IDS would detect missing fragments is due to a broken network. If a fragment is dropped or lost along the way to it's destination, the rest of the fragments in the fragment train will not be able to be completely reassembled. The receiving host will eventually time out and should issue an ICMP "Fragment Reassembly Time Exceeded" (type 11 code 1) error to the sender.

The second reason an IDS would detect missing fragments is because of an attacker purposely sending fragmented packets with missing pieces to a host to cause a denial of service. Every time a host receives fragmented packets, it waits a certain amount of time for all of the fragments to arrive. If the fragments don't arrive in that amount of time, an ICMP "Fragment Reassembly Time Exceeded" error message will be sent.

If an attacker wants to denial of service a machine, they can send many fragmented packets, leaving out some of the fragments needed. The recipient will wait for the other fragments to arrive and eventually time out. If the recipient receives so many fragments that it fills up it's buffers with incoming packets, it will start dropping other good packets it receives. This causes a denial of service for the machine.

Chances are, in the alerts seen here, a mixture of both reasons is occurring. While there are 265 alerts coming from 44 different external sources setting off this alert, there are only 29 unique internal destinations. The most any one internal destination receives is 74 packets; not enough to cause a full denial of service. The discarded packets in these alerts are probably due to a network error somewhere.

However, as shown below, there are 14,814 alerts from only 6 unique internal sources. Looking at the breakdown of where the packets are going to, we see that there are only 15 unique destination hosts, two of which receive over 6500 packets each from internal hosts. This indicates that some internal hosts have been compromised and are being used to denial of service these external machines.

Internal attacking hosts

Host	# alerts	# unique destinations
MY.NET.211.6	13,188	3
MY.NET.132.42	1,018	4
MY.NET.226.22	469	3
MY.NET.237.106	89	1
MY.NET.252.82	47	3
MY.NET.204.94	3	3

Top 5 attacked hosts

IP	# attacks	# attackers
198.247.231.42	6,963	2
216.111.123.20	6,656	2
172.181.116.159	584	1
172.181.251.235	200	1
172.180.246.250	152	1

Correlations: John Jenkinson noted in his GCIA paper seeing 14,601 alerts of this kind coming from only 18 unique sources. However, most of his alerts are coming from one

machine in Verio's network that he attributes to a broken network. Additionally, David Stewart reported seeing 4,309 alerts and also attributes it to non-malicious traffic.

However, there are known attacks against systems using this technique**.** Lance Spitzner discovered an attack against Checkpoint Firewall-1 machines where missing fragments could be used to denial of service the firewall. James Farrell details this attack in his paper, "IP Fragmentation Attacks on Checkpoint Firewalls", available in the SANS Reading Room.

Recommendations: Guarding against fragmentation denial of service attacks is not always easy. Configuring a firewall or router to drop all fragments would prevent these types of attacks, but may also prevent good traffic from getting through.

In the university, it appears that some internal machines have been compromised and are being used to denial of service external machines. These machines should be taken out of service and reloaded with current patches and security software to prevent another compromise. Further discussion of these hosts takes place in the compromised hosts section of this paper.

spp_http_decode: IIS Unicode attack detected

Total alerts reported: 5,892

External sources: 446 Unique external sources: 245

Internal sources: 5,446 Unique internal sources: 328

This alert is generated whenever the http_decode preprocessor detects a URL with Unicode characters that exploit vulnerabilities in the Microsoft IIS web server. These vulnerabilities allow an attacker to traverse the directory structure and execute commands directly on the server. The Code Red and Nimda worms also take advantage of these attacks to compromise IIS web servers and propagate. However, these Unicode characters can also appear in normal URLs and can cause false positives.

Since many of the worms that use IIS Unicode attacks to propagate scan for more machines to infect, internal machines setting off this alert could indicate an infected machine. Nevertheless, we have to be careful to determine whether or not a machine is actually infected, or if the alert it generated is due to a false positive. Unfortunately, the only foolproof way to tell this is to look at the actual URL that set the alert off, which is not shown in the logs.

The top 5 talkers for this alert are shown in the table below.

IP Address	# alerts	# unique destinations
MY.NET.242.250	424	58
MY.NET.97.172	180	4
MY.NET.236.66	176	11
MY.NET.207.34	172	13
MY.NET.112.204	157	3

The machine located at MY.NET.242.250 has the most alerts here and is one of the top 10 talkers of all the internal machines. Many of the alerts it generates are directed towards machines in the 211.233.0.0/16 networks, all located within Korea.

Given that there are so many alerts from this machine and that the destinations are spread out over a number of different machines, this is probably not a false positive. The fact that most of the destinations from MY.NET.242.250 are in Korea indicates that an internal user is either attacking Korean web servers, or this machine is infected with a worm that has chosen the 211.233.0.0/16 subnet to scan.

Correlations: Since the Code Red and Nimda worms were released, this signature is commonly seen in log files. Tod Beardsley noted seeing this alert 26,048 times in his analysis and attributed them to worm traffic. Additionally, there are numerous posts on many mailing lists detailing attacks that set off this alert being used by the Code Red and Nimda worms.

Rain Forest Puppy first explained the attacks detected by this alert in his advisory entitled “IIS %c1%1c bug”.

Recommendations: Every one of the internal IP addresses generating this alert, especially MY.NET.242.250, should be investigated further to see if they have been infected with a worm or have been compromised due to an IIS Unicode attack. If these machines are infected or have been compromised, their hard drives should be formatted, Windows reloaded and patched with the latest Windows and IIS patches. Additionally, tools like URLScan from Microsoft should be installed to help detect and prevent future attacks. If these machines do not need to have IIS installed at all, it should not even be loaded again.

Many routers and firewalls can now be configured to drop packets that contain URLs that set off this alert. Any routers or firewalls that provide this functionality on the university’s network should be configured to do so and alert an administrator when this is done.

High port 65535 tcp - possible Red Worm – traffic

Total alerts reported: 5,838

External sources: 2,122	Unique external sources: 166
Internal sources: 3,716	Unique internal sources: 59

This alert is generated whenever traffic to or from TCP port 65535 is seen. Port 65535 is used by the Adore worm, originally called the Red Worm, to open a backdoor to the infected system. The backdoor allows anyone connecting to it root access to the infected system.

By watching for traffic coming from or going to TCP port 65535, IDS analysts can be notified when a machine has been infected by the Adore worm and an attacker has connected to the backdoor. However, port 65535 is also an ephemeral port, and can be used in normal client/server connections. Therefore, careful attention must be paid to the traffic setting off the alerts to be sure that it is truly being generated by the backdoor from the worm and not normal traffic.

The top 5 internal machines that set off this alert are shown in the table below.

IP Address	# alerts	# unique destinations
MY.NET.207.214	770	1
MY.NET.204.74	735	1
MY.NET.201.146	589	2
MY.NET.202.226	550	1
MY.NET.243.238	344	1

Four of the top five internal machines that set off this alert are also in the top 10 alert talkers. It should be noted that these machines have a very small amount of unique destinations and are always sending data to port 65535 on the external machine. This means that these machines are either sending data on legitimate connections where their computer has chosen 65535 as the ephemeral port or they are connecting to an Adore backdoor on a remote machine. Unfortunately, since these machines did not generate any more relevant alerts and the packet payload is not available, it is not possible to tell.

Correlations: James Hoover details in his analysis one machine infected with the Adore worm and one false positive alert.

J. Anthony Dell has written an excellent explanation on the Adore worm and how it works. His paper is located at http://www.sans.org/rr/threats/mutation.php.

Recommendations: Any internal machine that traffic to port 65535 is seen travelling to should be investigated to see if they are infected with the Adore worm.

While it may not be possible to contact the owners of every machine that is seen contacting port 65535 externally, it should be feasible to contact the owners of the top talkers in this category. From there, it can be determined if malicious activity was taking place.

A stateful firewall should also be set up to block incoming and outgoing TCP connections to port 65535. The firewall should be set up to only block connection initiations to this port. This will prevent any internal or external users from connecting to a machine infected with the worm.

spp_http_decode: CGI Null Byte attack detected

Total alerts reported: 3,481
External sources: 24 Unique external sources: 19
Internal sources: 3,457 Unique internal sources: 121

This alert is generated when the Snort preprocessor http_decode detects a NULL Unicode byte in the URL. A Null Unicode byte is encoded as %00 in the URL and is used in attacks against CGI perl scripts on web servers to trick the CGI script into executing commands on the web server. However, this can be seen as a false positive if a form or CGI script passes this character in the URL.

Of all the 3,457 alerts generated by internal sources, none of them are for source port 80. In other words, every one of the 121 unique internal sources are connecting to port 80 and possibly attacking external machines. The following is a table of the top 5 internal talkers for this alert.

IP Address	# alerts	# unique external dest	Destination hosts
MY.NET.97.126	246	1	209.10.239.135
MY.NET.98.119	240	1	64.14.122.229
MY.NET.234.226	236	2	212.112.162.203, 212.112.171.37
MY.NET.97.67	211	1	216.241.219.14
MY.NET.237.82	187	1	209.10.239.135

As seen in the table above, each of the machines that set off this alert are going to only one or two unique destinations. It would be unusual to see an actual attack using a Null byte generate so many alerts to one host. Because of this, the alerts above indicate the IP addresses have a form or script that is constantly getting used and contains %00 in the URL.

In fact, one of the destinations that appears twice in the table above, 209.10.239.135, is the destination for 1,635 of the alerts. This IP address, which is owned by iFilm - an online streaming media website - is generating a false positive alert for every host that visits it.

Correlations: In his analysis, Joe Ellis details the “CGI Null Byte attack detected” alert and sees IP address 209.10.239.135 generating many alerts as well. As here, he concludes it is probably generating false positives.

Recommendations: The “CGI Null Byte attack detected” alert can generate a lot of false positives, depending on how much web traffic there is. Providing the “-cginull” option to the http_decode configuration line in Snort’s configuration file can disable this alert. This is recommended here as most, if not all, of the alerts are almost certainly false positives.

TFTP - External UDP connection to internal tftp server
Total alerts reported: 1,493
External sources: 0 Unique external sources: 0

Internal sources: 1,493 Unique internal sources: 5

This alert is generated whenever a connection is made from an internal source at port 69 (TFTP) to an external server. This is a concern because TFTP has no authentication and is used by many attackers and worms, including Code Red, to transfer files. A connection from an internal server to an external machine is especially a concern as it can mean the internal server has been compromised and files, such as rootkits, are being transferred to it.

There are only 5 unique, internal sources that set off this alert. They are detailed in the table below.

IP Address	# alerts	# Unique destinations
MY.NET.111.231	332	1
MY.NET.111.232	313	1
MY.NET.111.235	308	1
MY.NET.111.230	275	1
MY.NET.111.219	265	1

As the table shows, each of the hosts makes its connections for this alert to only 1 external destination. In fact, the external destination is the same for all hosts in this alert! The destination, 192.168.0.253, is a non-routable IP address used only for internal networks.

There could be a few things happening here. First, these machines could be spoofing the destination IP address and sending data off into the ethernet. This would be easy to do since UDP is connection-less and easier to spoof than TCP. However, there would be no point in sending data out onto the network where it had no where to go. This could possibly be used to flood the default gateway, since all of these hosts are on the same subnet, but there is not enough traffic to do this.

Most likely, 192.168.0.253 is actually on the internal network somewhere. TFTP has many legitimate uses for hosts, especially routers and diskless workstations. These machines use TFTP to download their configuration files. The five hosts that are setting off this alert are most likely routers or diskless workstations grabbing their configuration file from 192.168.0.253 and generating this false positive while doing so.

Correlations: Joe Ellis discusses this signature in his alert and states that it should never be allowed to internal servers from the Internet. However, Joe incorrectly misinterprets this alert as being a side effect of an FTP bounce attack.

The five hosts generating this alert are not the source for any other alerts during the time period analyzed from the university. Because of this, the alerts above are almost certainly false positives.

Recommendations: The hosts causing this alert should be located to verify that they are using TFTP for a legitimate purpose. Additionally, the host at 192.168.0.253 should be located to see if it is a valid server and what it is transferring to these hosts.

TFTP should never be allowed into or out of the internal network from external networks. It is recommended that a stateful firewall be placed between the internal network and any third-party networks (including the Internet) with a rule to block all TFTP traffic. Additionally, this signature should be kept in the IDS rules to alert whenever TFTP traffic from an external IP address is seen.

The signature should also be modified to not alert on TFTP traffic from 192.168.0.253 if it is found to be a valid host.

Scan logs analysis

A total of 258,798 scans were detected and logged. The following table shows the scans ordered by number of occurrence. The table also breaks down the number of scans from internal and external sources.

# Scans	Internal Sources	External Sources	Scan Type
150,030	114,284	35,749	SYN
106,051	106,050	1	UDP
1,309	3	1,306	NULL
486	264	222	FIN
392	4	388	NOACK
212	2	210	INVALIDACK
149	0	149	VECNA
128	21	107	UNKNOWN
14	0	14	XMAS
11	0	11	NMAPID
8	0	8	SPAU
5	0	5	FULLXMAS
3	0	3	SYNFIN

Top 10 internal scan sources

# Scans	IP Address
111,366	MY.NET.223.78
20,414	MY.NET.70.176
14,995	MY.NET.87.44
6,185	MY.NET.98.31
5,377	MY.NET.97.136
5,349	MY.NET.242.174
4,198	MY.NET.97.110
3,799	MY.NET.97.67
3,058	MY.NET.98.150
2,794	MY.NET.97.35

Top 10 external scan sources

# Scans	IP Address
5,380	66.134.226.37
2,521	80.14.80.158
2,422	64.156.31.70
1,824	63.78.224.166
1,673	210.178.9.1
1,517	206.167.165.56
1,503	213.73.142.100
1,254	61.242.90.229
720	218.155.10.85
702	12.239.36.3

Top 10 ports scanned for

# Scans	Port	Description
120,124	443	HTTPS over SSL
21,996	22,321	
20,690	6,257	WinMX file sharing app
16,370	137	NetBIOS Name Service
12,551	7,674	iMQ SSL Tunnel
11,152	445	Win2k+ SMB
10,370	27,005	FlexLM (1-10)
7,468	80	HTTP
1,829	21	FTP
1,390	0	

Interesting SYN Scans

SYN and UDP scans were detected the most of all of the scans and there are some interesting trends within these scans.

SYN scans occur when an attacker sends a TCP packet with just the SYN flag set to a specific port or ports on a victim. Per RFC 793 guidelines, the victim should respond with a SYN/ACK packet if the service is available, or a RST/ACK if the service is not. This helps an attacker quickly determine what hosts have a specific service running.

Most of the SYN scans were from internal sources. In fact, 111,366 of the 150,030 SYN scans were from one source: MY.NET.223.78. This host scans a large amount of IP addresses looking for any machine with TCP port 443 open. TCP port 443 is the default port that HTTPS over SSL listens on. In July 2002, a number of remotely exploitable vulnerabilities were found in the OpenSSL libraries that provide the SSL layer for many applications, including the Apache web server. These vulnerabilities are detailed in CERT advisory CA-2002-23.

In September 2002, a self-propagating worm was released which would exploit this vulnerability in unpatched Apache servers. This worm is detailed in CERT Advisory CA-2002-27. However, MY.NET.223.78 is probably not infected with this worm. According to the CERT advisory, the worm will first scan for vulnerable systems on port 80, then connect to port 443 when it finds one. Although MY.NET.223.78 scans for port 80, it does so only minimally and the majority of its scans are for port 443.

Therefore, MY.NET.223.78 is most likely being used to scan for any servers that have port 443 open. Attackers like to create lists of machines running specific services so when an exploit is released in the future, they already have a list of machines that are vulnerable. MY.NET.223.78 is probably being used to create one of these lists.

Interesting UDP Scans

UDP scans occur whenever an attacker is searching for services listening on UDP ports. To find out if a machine has a UDP service open, the attacker will send a UDP packet to the sought after port. If the attacker receives an ICMP Port Unreachable

message, they know the port is closed. If the attacker does not receive a response, as none is necessarily required with a connectionless protocol such as UDP, they assume the port is open.

This technique causes many false positives as many firewalls will drop UDP packets for services they do not allow. When this occurs, the scanner will never receive a response and assume the port is opened.

Many of the UDP scans detected were from internal hosts as well. The top ports scanned for with UDP scans were ports 22321, 6257, 137 and 7674. Internal machines scanned for all of these ports.

Port 22321

Scanning for UDP port 22321 was the highest UDP port scanned for, with 21,996 separate scans. I was not able to find what service listened on UDP port 22321, but the Windows trojan backdoor Dobol listens on TCP port 22321.

Greg Schmidt posted to the Incidents mailing list in September 2002 that he saw many scans from his university students for this port. The Dobol backdoor was suggested as the reason for the scans, but it was pointed out that this uses TCP not UDP. Unfortunately, searching the Internet did not turn up any more information about this.

Looking at the packets from the scan for this port, it is evident that packet crafting is present. As shown below in one of the logged scans, the source port is the same as the destination port. This rarely occurs with legitimate traffic and is a definite indication a tool is creating the packet from scratch.

Feb 15 00:21:13 MY.NET.97.212:22321 -> 203.247.198.108:22321 UDP

Port 6257

The WinMX peer to peer file-sharing program uses UDP port 6257. Many peer to peer file sharing programs work by talking to their peers to find the files they are looking for. Because they connect to many different peers at once, the traffic can set off portscan alerts. This is what is occurring here.

As shown below, much of the traffic to port 6257 is also coming from source port 6257. While this usually indicates packet crafting, this is not the case here. A post to the alt.music.mp3.winmx newsgroup by Dolphy on August 18, 2002 concerning what firewall rules to open for WinMX to work indicates that he regularly sees traffic from WinMX with reflexive ports.

Feb 15 04:00:33 MY.NET.70.176:6257 -> 68.7.35.15:6257 UDP

Port 137

UDP port 137 is used by the NetBT Name Service on Windows machines to resolve NetBIOS names and provide information on the different services that are available on a Windows machine, including what file shares are open. While this traffic is normal on Windows networks, scans for port 137 mean an attacker or virus is looking for open shares to attack.

Traffic for UDP port 137 will sometimes be seen with reflexive ports, as shown below. This occurs when a user is running the Windows utility "nbtstat" with the "-a" or "-A" option. Running "nbtstat" with these options will dump the remote machine's NetBIOS name table and display information about the computer and whether some services, such as file shares, are present.

Feb 19 11:37:39 MY.NET.97.112:137 -> 208.86.211.151:137 UDP

In all of the UDP scans, only one source, MY.NET.97.112, shows this signature. The scans from this machine occur sporadically over the time period analyzed, so this is probably a user that is manually looking for open file shares on machines that they come across.

The rest of the traffic comes from an ephemeral port, like the scan shown below. When scans like this are seen, it usually means a virus has infected a computer and is scanning for more open file shares to infect.

Feb 15 09:00:08 MY.NET.97.110:1030 -> 61.147.211.132:137 UDP

There are 57 internal servers logged scanning for UDP port 137. The top talker of this group, MY.NET.97.110, logged 4,071 scans for port 137. Additionally, this host caused a number of "IIS Unicode Attack detected" and "SMB Name Wildcard" alerts to be generated. Therefore, like many of the other internal machines scanning for port 137, it is infected with a worm.

Port 7674

Port 7674 is described as belonging to iMQ SSL Tunnel, a component of the iPlanet Message Queue for Java. However, this is probably not what is being scanned for here. When looking at the packets that are detected by the UDP scan, we see that, like the scans for 22321, they have reflexive ports.

Feb 15 00:21:51 MY.NET.97.212:7674 -> 211.232.199.249:7674 UDP

In fact, it is interesting to note that every one of the machines that scanned for UDP port 22321, also scanned for UDP port 7674! This helps us infer that the same program is scanning for both ports. We can probably also hypothesize what is going on.

As seen in previous detects, university networks are full of peer to peer file sharing network traffic. What could be occurring here is that a new peer to peer file-sharing

program is using these two ports to transfer files or information. Since peer to peer programs talk to each other to find out where files are, this would look like a scan, as seen with the WinMX scans above.

Khan Rohail posted to the Security-Basics mailing list on February 13, 2002 that he saw UDP traffic on ports 22321 and 7764 going to and coming from students systems in his university. This correlates our theory that the same tool is scanning for both ports.

Additional Scans Detected

The rest of the scans detected work by manipulating the flags in TCP packets. In doing so, the scanner can cause the victim machine to respond in a variety of ways to indicate whether or port is open or not. Additionally, the scanner may be able to trick firewalls into thinking the packet is part of a valid connection and allow the packet through.

The following table, taken from Joe Ellis' GCIA analysis paper, shows what flags must be set in order for these scans to be detected. Joe got this from Christof Voemel's paper.

VECNA	One of the following: P, U, PU, FP, FU
NULL	None of SFRPAU
UNKNOWN	See spp_portscan.c source code
NOACK	A flag is missing
INVALIDACK	ACK set, not 'normal', no SPAU or FULLXMAS
FIN	F flag
XMAS	FPU flags
SPAU	SPAU flags
SYNFIN	SF flags
NMAPID	SFPU flags
FULLXMAS	SFRPAU flags

Since there are no written RFCs that explain how an IP stack should respond when it receives these unusual combinations of flags, each operating system has done their own thing. Because of this, fingerprint scanners are available which will send machines odd packets in the hope that the program will be able to tell what operating system is responding by the way it formats the packets.

However, false positives of these scans are also seen within the logs. For example, as has already been shown, much of the traffic in the logs is from peer to peer file-sharing programs. In the scans detected, 510 of the scan alerts were from peer to peer programs.

Due to the way that some of the peer to peer programs communicate, the portscan preprocessor may interpret them as portscans. Richard Bejtlich noted in a post to the Intrusions mailing list that some Gnutella communications (port 6346) are interpreted by Snort as Vecna scans. This is also seen in the scan logs from the university, as shown below. **Shane Huntley** also noted this about KaZaa communications in his analyses as well.

Feb 15 07:15:00 148.64.29.21:1026 -> MY.NET.202.50:6346 VECNA ****P***

OOS packet analysis

The Out-Of-Spec (OOS) log files are packets detected that have unusual or illegal TCP flag combinations. These include packets with the ACK flag missing when it is required or the Type of Service reserved bits set.

The following is a list of the top 10 OOS talkers.

# packets	IP Address	Reverse lookup
445	148.64.169.5	vsat-148-64-169-5.c005.g4.mrt.starband.net
368	148.63.130.172	vsat-148-63-130-172.c189.t7.mrt.starband.net
347	65.214.38.10	ghost.directhit.com
300	68.164.35.154	h-68-164-35-154.NYCMNY83.covad.net
203	213.98.16.183	213-98-16-183.uc.nombres.ttd.es
203	210.253.215.113	nttfad3-113.246.ne.jp
155	200.163.200.5	3-005.ctame700-1.telepar.net.br
149	61.114.222.241	h222241.ppp.asahi-net.or.jp
106	212.86.100.68	hunter.rbone.ci.net.ua
95	216.95.201.18	smtp8.jsuati.com

The following analysis of some OOS packets will look at three hosts, each logged as out of spec for different reasons.

OOS packet #1 - 148.63.130.172
Number of unique destinations: 1
Number of unique destination ports: 1
Reason for OOS: No ACK flag – PSH flag by itself – No ACK number

This host is the second highest OOS talker with 368 packets logged. However, all of the packets for this host are directed to one internal IP address on one port: MY.NET.229.58 on port 3676. By looking at the data within the packet, we can see that these packets are part of a conversation using KaZaa.

```
02/15-10:51:29.267008 148.63.130.172:1582 -> MY.NET.229.58:3676
TCP TTL:115 TOS:0x0 ID:48733 IpLen:20 DgmLen:440 DF
****P*** Seq: 0x408EB80A  Ack: 0x0  Win: 0x2000  TcpLen: 20
47 45 54 20 2F 2E 68 61 73 68 3D 32 32 36 33 62  GET /.hash=2263b
39 63 32 31 65 37 65 30 62 38 34 39 34 35 32 61  9c21e7e0b849452a
38 32 64 38 66 34 62 61 65 64 66 63 62 65 38 38  82d8f4baedfcbe88
62 65 36 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F  be6 HTTP/1.1..Ho
73 74 3A 20 31 33 30 2E 38 35 2E 32 32 39 2E 35  st: MY.NET.229.5
38 3A 33 36 37 36 0D 0A 55 73 65 72 41 67 65 6E  8:3676..UserAgen
74 3A 20 4B 61 7A 61 61 43 6C 69 65 6E 74 20 4E  t: KazaaClient N
6F 76 20 20 33 20 32 30 30 32 20 32 30 3A 32 39  ov  3 2002 20:29
3A 30 33 0D 0A 58 2D 4B 61 7A 61 61 2D 55 73 65  :03..X-Kazaa-Use
72 6E 61 6D 65 3A 20 66 69 74 69 6E 64 72 67 6E  rname: 123456789
```

```
0D 0A 58 2D 4B 61 7A 61 61 2D 4E 65 74 77 6F 72  ..X-Kazaa-Networ
6B 3A 20 4B 61 5A 61 41 0D 0A 58 2D 4B 61 7A 61  k: KaZaA..X-Kaza
… (cut for brevity)
```

This packet is marked as OOS because it has an illegal combination of flags: only the PSH flag is set. During a TCP conversation, the ACK flag will always be set in order to acknowledge the previous packet's data. The only time the ACK flag is not set is during the initial SYN packet - every time after that the ACK flag should always be set. We know this is not part of an initial TCP connection because the SYN flag is not set.

Apparently, some peer to peer file-sharing programs will send data with only the PSH flag set. In a paper entitled "Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts", the authors provide a figure showing the exact TCP sequence that Gnutella clients and hosts communicate (Howe 11). During a Gnutella Handshake and Gnutella Packet-Pair Estimate, the peers will send TCP traffic with only the PSH flag set. This causes packets to be marked as OOS and alerted as scans. This can be seen in the logs as well because the traffic between these two hosts generated 30 Vecna scan alerts.

Unfortunately, peer to peer file sharing programs are commonly seen on networks today and generate many false positives. In fact, out of the 7,473 OOS packets logged within the time period analyzed, at least 1,810 packets were from peer to peer file-sharing programs.

OOS Packet #2 - 65.214.38.10
Number of unique destinations: 33
Number of unique destination ports: 1
Reason for OOS: TCP flag byte reserved bits set

```
02/18-21:40:34.048027 65.214.38.10:39446 -> MY.NET.70.231:80
TCP TTL:46 TOS:0x0 ID:14199 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xD11B5FA3  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 251947220 0 NOP WS: 0
```

The host here was marked as OOS because the reserved bits in the TCP flag byte (bits 6 and 7) were set. When TCP was first implemented, these bits were set to be reserved and should always have been set to 0.

However, the IETF released RFC 2481, later superceded by RFC 3168, which allows for the use of these bits for Explicit Congestion Notification (ECN). ECN provides a way to reduce congestion on a network by having hosts reduce the amount of traffic sent. As shown below, bit 7 is the Congestion Window Reduced bit (CWR) which notifies senders to send less data. Bit 6 is known as the ECN echo flag (ECN) and is set when congestion is experienced.

7	6	5	4	3	2	1	0
CWR	ECN	URG	ACK	PSH	RST	SYN	FIN

When these bits are set, however, packets could be mistaken as packets coming from Queso, an active OS fingerprinting program. Toby Miller verifies this in his paper "ECN and it's impact on Intrusion Detection" located on the SANS web site. The university's logs indicate this as well because 65.214.38.10 generates 32 Queso alerts as well.

However, 65.214.38.10 could still be an attacking machine. DShield's database shows a large number of attacks from this IP address directed at a number of different ports. Considering that this host is part of the Ask Jeeves search engine network, we should expect to never see initiated sessions from this host, as is the case here.

OOS Packet #3 - 24.165.17.183
Number of OOS packets: 10
Number of unique destinations: 1 Number of unique destination ports: 2
Reason for OOS: Illegal flag combinations

Like many of the other packets within the OOS files, host 24.165.17.183 talks to only one destination, MY.NET.240.178, on a small number of ports. In the conversations here, 24.165.17.183 talks to TCP ports 1191 and 6699. Port 6699 is used with Napster and WinMX file sharing, but I could not find any known application that used port 1191.

These packets were marked as OOS because they each had illegal flag combinations in each packet. An example packet is shown below.

```
02/16-00:24:00.082676 24.165.17.183:1191 -> MY.NET.240.178:6699
TCP TTL:111 TOS:0x0 ID:33959 IpLen:20 DgmLen:76 DF
****PRSF Seq: 0xD38215  Ack: 0x8D67  Win: 0x5018  TcpLen: 12
00 00 ED 4C B5 1B D9 62 01 2B 70 7E 66 B8 E3 54  ...L...b.+p~f..T
5E 2C 02 35 05 DD 8E 06 77 2E 7A 24 59 76 2F A1  ^,.5....w.z$Yv/.
E6 78 A3 97                                      .x..
```

In this packet, the PSH, RST, SYN and FIN flags are all set at the same time. This is an illegal combination of flags and should never occur. The following table lists all of the flags set in each of the OOS packets from 24.165.17.183.

Packet #	CWR	ECN	URG	ACK	PSH	RST	SYN	FIN
1					X	X	X	X
2				X	X		X	X
3								
4								
5	X	X		X			X	
6			X		X	X	X	X
7	X	X	X			X		X
8				X			X	X
9								
10				X	X	X	X	X

In addition to the illegal flags, every packet had different and widely varying sequence numbers, datagram lengths and IP Identification numbers.

Every one of the flag combinations in the packets above is illegal and will never occur in normal traffic. Since these are illegal combinations of flags, they were never planned for when the RFCs for IP and TCP were created. Therefore, every operating system handles them in its own way by responding in different ways and setting different options in the reply packets.

The attacking host here, 24.165.17.183, is using illegal flags to discover the operating system of MY.NET.240.178. By sending packets with illegal flag combinations to a remote system and analyzing the responses, one can determine the remote operating system. This is known as active OS fingerprinting and a number of programs are available that provide this functionality, including Queso and nmap.

Registration Information

The following is the registration information for the top 5 external alert talkers.

169.232.84.146
Reverse DNS: `s84-146.resnet.ucla.edu.`
WHOIS Information: whois.arin.net

```
University of California, Office of the President UCNET-BLK
(NET-169-228-0-0-1) 169.228.0.0 - 169.237.255.255
University of California, Los Angeles UCLANET4
(NET-169-232-0-0-1) 169.232.0.0 - 169.232.255.255
```

12.35.158.199
Reverse DNS: `unknown`
WHOIS Information: whois.arin.net

```
AT&T WorldNet Services ATT
(NET-12-0-0-0-1) 12.0.0.0 - 12.255.255.255
Mckenzie Tankline MCTAN656-158-192
(NET-12-35-158-192-1) 12.35.158.192 - 12.35.158.207
```

The next three hosts: 212.179.123.163, 212.179.88.96 and 212.179.105.210 all produced the same contact information. For brevity, this is only listed once.

212.179.123.163
Revesre DNS: `cablep-179-123-163.cablep.bezeqint.net.`
WHOIS Information: whois.ripe.net

```
inetnum: 212.179.100.0 - 212.179.124.255
netname: CABLES-CONNECTION
descr: CABLES-CUSTOMERS-CONNECTION
country: IL
admin-c: YK76-RIPE tech-c:
BHT2-RIPE
```

```
status: ASSIGNED PA
remarks: please send ABUSE complains to abuse@bezeqint.net mnt-by:
AS8551-MNT mnt-lower: AS8551-MNT
notify: hostmaster@bezeqint.net
changed: hostmaster@bezeqint.net 20021029
source: RIPE

route: 212.179.64.0/18
descr: ISDN Net Ltd.
origin: AS8551
notify: hostmaster@bezeqint.net
mnt-by: AS8551-MNT
changed: hostmaster@bezeqint.net 20020618
source: RIPE

role: BEZEQINT HOSTMASTERS TEAM
address: bezeq-international
address: 40 hashacham
address: petach tikva 49170 Israel
phone: +972 1 800800110
fax-no: +972 3 9203033
e-mail: hostmaster@bezeqint.net
admin-c: YK76-RIPE
tech-c: MR916-RIPE
nic-hdl: BHT2-RIPE
remarks: Please Send Spam and Abuse ONLY to abuse@bezeqint.net
mnt-by: AS8551-MNT changed: hostmaster@bezeqint.net 20021029
changed: hostmaster@bezeqint.net 20030204
source: RIPE

person: Yuval Keinan
address: bezeq-international
address: 40 hashacham
address: petach tikva 49170 Israel
phone: +972 1 800800110
fax-no: +972 3 9203033
e-mail: hostmaster@bezeqint.net
mnt-by: AS8551-MNT
nic-hdl: YK76-RIPE
changed: hostmaster@bezeqint.net 20021215
changed: hostmaster@bezeqint.net 20030204
source: RIPE
```

212.179.88.96

Reverse DNS: `bzq-179-88-96.cablep.bezeqint.net.`

WHOIS Information: whois.ripe.net

```
inetnum: 212.179.80.0 - 212.179.94.255
netname: CABLES-CONNECTION
descr: CABLES-CUSTOMERS-CONNECTION
country: IL
admin-c: YK76-RIPE
tech-c: BHT2-RIPE
status: ASSIGNED PA
remarks: please send ABUSE complains to abuse@bezeqint.net
```

```
mnt-by: AS8551-MNT
mnt-lower: AS8551-MNT
notify: hostmaster@bezeqint.net
changed: hostmaster@bezeqint.net 20021029
source: RIPE
```

212.179.105.210
Reverse DNS: `cablep-179-105-210.cablep.bezeqint.net.`
WHOIS Information: whois.ripe.net

```
inetnum: 212.179.100.0 - 212.179.124.255
netname: CABLES-CONNECTION
descr: CABLES-CUSTOMERS-CONNECTION
country: IL
admin-c: YK76-RIPE tech-c:
BHT2-RIPE
status: ASSIGNED PA
remarks: please send ABUSE complains to abuse@bezeqint.net mnt-by:
AS8551-MNT mnt-lower: AS8551-MNT
notify: hostmaster@bezeqint.net
changed: hostmaster@bezeqint.net 20021029
source: RIPE
```

Additional Defensive Recommendations

While defensive recommendations are placed throughout this analysis, there are a few recommendations that can be made which will help in overall detection.

Peer to peer (P2P) file sharing traffic is rampant within the university's network. By allowing P2P software to run, a number of issues arise. First, the P2P networks create an unsecured, unprotected tunnel into the network where viruses, disguised as normal files, can enter when unsuspecting users download them. Once these viruses get in and infect the P2P client, they will propagate to other internal machines.

Additionally, many legal issues arise when users are allowed to download illegal files, such as copyrighted MP3s. Furthermore, as seen in the analysis, the P2P traffic creates many false positive alerts that waste an analyst's time.

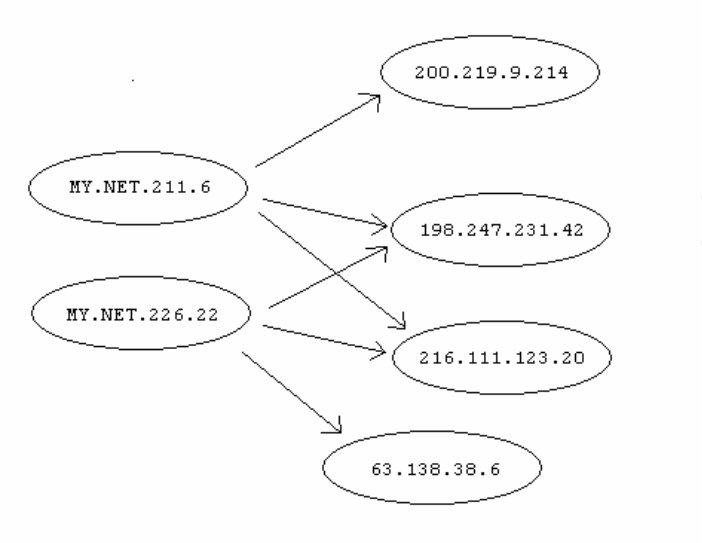
A stateful firewall should be put into place to block all incoming and outgoing connections to the ports P2P software uses, such as TCP 1214 (KaZaa), TCP 6346 (Gnutella) and TCP 4662 (eDonkey). This is not fool proof, since users can change the port the software listens on, but this will still block many connections. A policy banning the use of all P2P software should also be drafted.

There is also a lot of activity on the Microsoft NetBIOS and file sharing ports (ports 135-139, 445) from external machines. Traffic from these ports should never be allowed into a network from external machines. Firewall rules should be put into place, which block all traffic into and out of the internal network on these ports.

Compromised Hosts

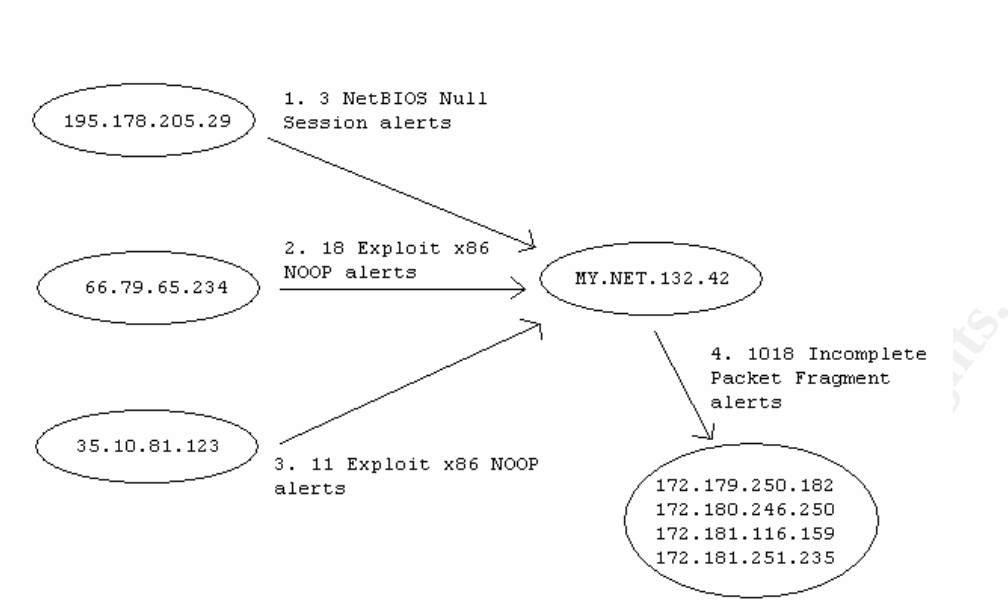
There is evidence a number of internal hosts have been compromised and are being used as distributed denial of service (DDoS) agents. During the analysis of the "Incomplete Packet Fragments Discarded" alert, it was found that a few internal sources were generating this alert many times to a small number of external sources. Since this alert can be generated when a denial of service attack takes place, chances are high that these hosts have been compromised and are being used in a DDoS network.

Two of the top internal hosts for the alert above, MY.NET.211.6 and MY.NET.226.22, are DoS'ing a number of external hosts. The following link graph shows which hosts the two internal machines generated incomplete fragment alerts to.



Unfortunately, none of the alerts for the two internal machines give any indication of who may have compromised them, so it must have taken place before the time period analyzed.

However, another internal machine that produces a large number of incomplete fragment alerts, MY.NET.132.42, does have alerts that show it being compromised. The following link graph shows the sequence of alerts generated to and from this host.



In the sequence of alerts above, MY.NET.132.42 is first connected to with a Null Session by 195.178.205.29. While a Null Session can occur in normal network traffic, you should never see one coming from an external machine. Next, 18 "Exploit x86 NOOP" alerts are generated from 66.79.65.234 followed by 11 of the same alerts from 35.10.81.123. These alerts indicate a buffer overflow exploit is being run against MY.NET.132.42 and that there is a high chance that either, or more likely both, of the attacking machines have compromised it. After the NOOP alerts occur, 1018 incomplete packet fragments are generated from MY.NET.132.42 to the four hosts listed.

Each of the three internal machines described here is being used as DDoS zombies in attacks against external machines. They should all be taken off-line and forensically examined to find out how they were compromised. If the DDoS software can be obtained from the forensic analysis, it should be analyzed and Snort signatures created so other zombie hosts in the network can be detected.

Analysis Process

The analysis of the log files was performed on a Linux notebook using command line tools and perl. To analyze the alert logs, some perl scripts were written to pull out specific data, such as the alert name and source and destination IP addresses. Once these were pulled out, UNIX command line tools including awk, grep, sort and uniq were used to grab statistical information for analysis.

For the scan and OOS logs, the command line tools noted above were used. Since these logs were in a more regular format, these tools provided quick and easy ways to pull the data needed.

Throughout the entire analysis, running "sort | uniq –c | sort –nr" helped immensely when trying to find the top talkers for a number of different reasons. For example, whenever a specific alert was being analyzed, these commands would be used to pull the top talkers or top destinations so any patterns could be detected. Without these commands, the data would have had to be imported into a spreadsheet or database and analysis done from that, which would have taken considerably longer.

References

Alexander, Bryce. "Intrusion Detection FAQ: Port 137 Scan." 10 May 2000. <http://www.sans.org/resources/idfaq/port_137.php>.

Baird, Scott. "Intrusion Detection In Depth GCIA Practical Assignment." <http://www.giac.org/practical/Scott_Baird_GCIA.doc>.

Beardsley, Tod. "Intrusion Detection and Analysis: Theory, Techniques, and Tools." <www.giac.org/practical/Tod_Beardsley_GCIA.doc>.

Bejtlich, Richard. "'venca' history." 6 Feb 2002. Incidents mailing list. <http://www.incidents.org/archives/intrusions/msg03111.html>.

"CERT® Advisory CA-2002-23 Multiple Vulnerabilities In OpenSSL." CERT. 11 Oct 2002. <http://www.cert.org/advisories/CA-2002-23.html>.

"CERT® Advisory CA-2002-27 Apache/mod_ssl Worm." CERT. 11 Oct 2002. <http://www.cert.org/advisories/CA-2002-27.html>.

Coyle, Brian. "GCIA Practical V3.1." <http://www.giac.org/practical/GCIA/Brian_Coyle_GCIA.pdf>.

Dell, J. Anthony. "Adore Worm – Another Mutation." 6 April 2001. <http://www.sans.org/rr/threats/mutation.php>.

Dolphy. "Re: Winmx *source* port 6257 - Valid?" Online posting. 18 Aug 2002. <news:alt.music.mp3.winmx>.

Ellis, Joe. "GCIA Practical Assignment, v3.0." <http://www.giac.org/practical/Joe_Ellis_GCIA.doc>.

"Exploitation of Unprotected Windows Networking Shares." CERT. 7 April 2000. <http://www.cert.org/incident_notes/IN-2000-02.html>.

Farrell, James. "IP Fragmentation Attacks on Checkpoint Firewalls." 3 April 2001. <http://www.sans.org/rr/firewall/frag_attacks.php>.

Hoover, James. "SANS GCIA Practical." <http://www.giac.org/practical/James_Hoover_GCIA.doc>.

Howe, Anthony J, and Dr. Mantis Cheng. "Napster and Gnutella: a Comparison of two Popular Peer-to-Peer Protocols". 28 Feb 2002. <www.cs.washington.edu/homes/tzoompy/publications/mm_systems_journal/2002/mm.pdf>.

Jenkinson, John. "GCIA Practical." <http://www.giac.org/practical/John_Jenkinson_GCIA.doc>.

McDonnell, Michael. "Intrusion Detection In Depth GCIA Practical Assignment." <http://www.giac.org/practical/Michael_McDonnell_GCIA.doc>.

Miller, Toby. "ECN and it's impact on Intrusion Detection." <http://www.sans.org/y2k/ecn.htm>.

Rain Forest Puppy. "IIS %c1%1c bug." 28 Feb 2001. <http://www.wiretrip.net/rfp/p/doc.asp/i2/d57.htm>.

Ramakrishnan, K. "RFC 2481: A Proposal to add Explicit Congestion Notification (ECN) to IP." Jan 1999. <http://www.ietf.org/rfc/rfc2481.txt?number=2841>.

Rohail, Khan. "UDP Traffic on port 22321 AND 7674." 13 Feb 2003. Security-Basics mailing list. <http://www.securityfocus.com/archive/105/311770>.

"rpc-32771 (330): RPC bind service on improper port." ISS X-Force Database. 4 June 1997. <http://www.iss.net/security_center/static/330.php>.

Schmidt, Greg. "UDP port 22321." 9 Sept 2002. Incidents mailing list. 2 April 2003. <http://cert.uni-stuttgart.de/archive/incidents/2002/09/msg00054.html>.

Spitzner, Lance. "FW-1 IP Fragmentation Vulnerability." BugTraq. 5 June 2000. <http://www.securityfocus.com/archive/1/63478>.

Steward, David. "GCIA Practical Assignment." <http://www.giac.org/practical/david_stewart_gcia.doc>.

Voemel, Christof. "Christof Voemel SANS Intrusion Detection Practical." <http://www.giac.org/practical/Christof_Voemel_GCIA.txt>.

Worman, Mike. "Intrusion Detection Practical Assignment." <http://www.giac.org/GCIA/Mike_Worman_GCIA.doc>.