# GIAC TRAINING AND CERTIFICATION

## TRACK 3 – INTRUSION DETECTION IN-DEPTH
## GCIA PRACTICAL ASSIGNMENT VERSION 3.3

DOCUMENT VERSION 2.0

SUBMITTED BY

JOHNNY WONG

## Abstract

This practical assignment is submitted as part of the GCIA (GIAC Certified Intrusion Analyst) certification process. This paper consists of 3 parts. The first part discusses the threat of corporate remote access services. The second part describes and analyses 3 network detects. The final part analyses five days worth of logs collected by an unnamed University.

# Part 1:Describe the state of Intrusion Detection

# The threat of corporate remote access services

## 1. Introduction

An article by Kevin Tolly in NetworldWorldFusion titled *"Always on programs pose an always on threat"*[1] caught my attention while I was looking for a subject to write on for this practical. For most organizations, the implementation of firewalls for perimeter defense, VPNs for secure remote access, IDSes, content filters, anti-virus, personal firewalls etc. would be sufficient to counter the external threat from the Internet. But as we have heard countless times before, most network compromise stemmed from employees.

In this paper, without regarding non-business programs like Kazaa, Morpheus, and Trojans like SubSeven, BackOrifice, I will look at legitimate programs that allow users to access their office desktops remotely from anywhere in the Internet. Bear in mind these are legitimate programs, which users can easily download and install the client because they want the flexibility. One such example, which I shall attempt is study, is Expertcity's GoToMyPC.

Traditional corporate remote access was implemented using VPNs, and even tools like Symantec's PC Anywhere. However, network managers/administrators were faced with issues like distribution of client software/updates, inability to scale and firewall configuration issues. GoToMyPC helps to solve these issues to a certain extent. One obvious advantage is that no software is required on the user's PC, just a Java-enabled Web browser would do. As taken from www.gotomypc.com[2], GoToMyPC resides as an always-on program on the desktop, communicating with the GoToMyPC server by means of a "heart-beat" communication. A user who wishes to access his PC would log on to the GoToMyPC service, authenticates himself, and voila! gains control of his remote desktop. During the process, there were no incoming connection requests to the desktop, instead the communications were initiated outwards. Most organizations' firewalls permit outbound access, making GoToMyPC easily deployable.

From this scenario, we look at some of the security implications:

a. The GoToMyPC server acted as the broker throughout the session between the user and the remote desktop. Wouldn't the server be able to deduce when the user is in office, the amount of activity on the desktop, the working habits etc. (as pointed out by Tolly)?
b. With the ease of obtaining the software, how do we detect whether any user within the enterprise has installed the software? This would require inspection of the outgoing traffic, which I will attempt to capture later. We might need to amend the corporate firewall policy to block such traffic, if possible.
c. How do we trust a third-party (i.e. Expertcity) that all the transactions were

not recorded? Since all the traffic has to go through their servers.

d. Although the sessions between the user, remote desktop and the GoToMyPC server are encrypted using AES, it may still be possible for an attacker to eavesdrop and look for session keys.

e. When a remote desktop (with GoToMyPC running) is accessed from a PC, a cookie is created which is used to track traffic patterns and retrieve registration information. The cookie holds a unique number generated at the time of registration, but does not contain any personally identifiable information or passwords. According to Expertcity, the cookie cannot be used by an attacker to access another user's account. However, if an attacker is able to locate the active cookie, can he actually hijack the session?

f. Lastly, the desktop with the GoToMyPC software loaded would most likely be located and trusted in the enterprise network. It would have access to all the network resources available. Wouldn't it be a scary thought if somehow, the access codes and passwords were compromised?

My deepest concern would be the ease of obtaining and running this software **without** the knowledge of the organization. Imagine an ignorant employee accessing his office desktop from shared public PC (e.g. Internet café) and failing to disconnect at the end of a session. The risk is too great to ignore. In order to understand the implications, we need to examine the software, what it does, how it does it and if possible, are there any loopholes in the program?

## 2. The GoToMyPC solution

The GoToMyPC system is a hosted service comprising of four components:

**Computer (Client):** A small footprint server (Servlet) is installed on the computer to be accessed. Typically, this is a home or office PC with always-on Internet connectivity. This server registers and authenticates itself with the GoToMyPC broker server.

**Browser (User):** The remote or mobile user launches a Web browser, visits the secure GoToMyPC website, enters a username/password and clicks a "Connect" button for the desired computer, sending an SSL-authenticated and encrypted request to the broker.

**Broker (Server):** The broker is a matchmaker that listens for connection requests and maps them to registered computers. When a match occurs, the broker assigns the session a communication server. Next, the client viewer – a tiny session-specific executable – is automatically loaded by the browser's Java Virtual Machine. The GoToMyPC viewer runs on any computer with a Java-enabled browser, including wireless devices.

**Communication Servers:** The communication server is an intermediate system that relays an opaque and highly compressed encrypted stream from client to server for the duration of each GoToMyPC session.

The following diagram was extracted from the GoToMyPC's technical document on security.

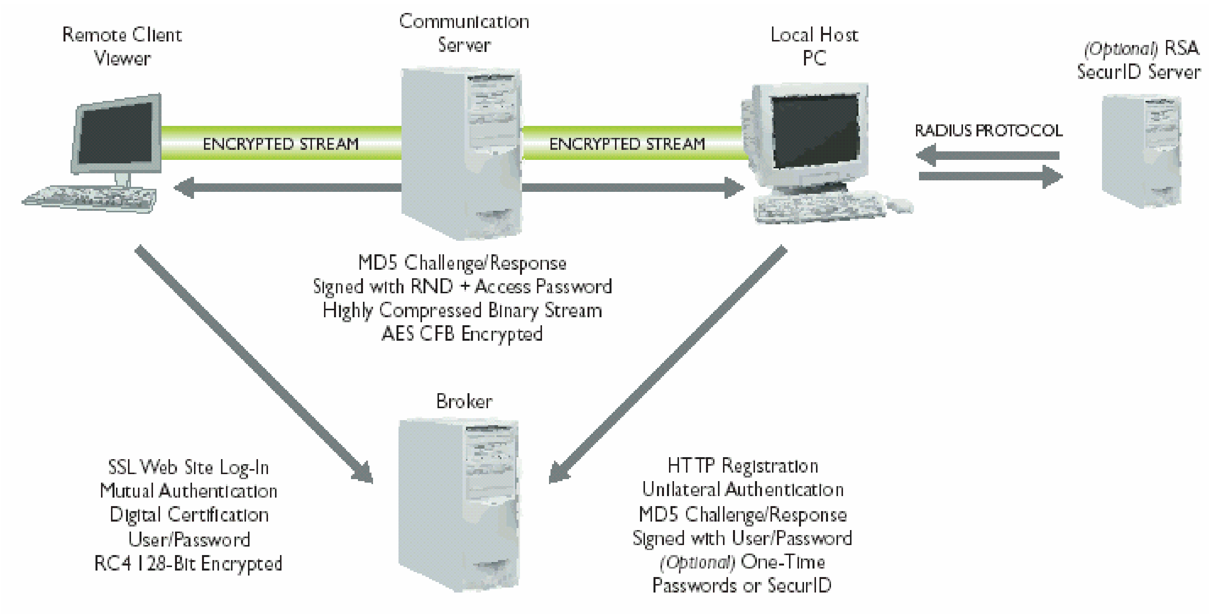



**Figure 1** GoToMyPC's Security Architecture

Expertcity has put in place a few security measures[3] to gain customers' confidence in the GoToMyPC solution. For instance, all GoToMyPC web, application, communication and database servers are hosted in a highly secured data center. Physical access to the servers is restricted. The entire site sits in a locked cage that is monitored by cameras. Expertcity's network operations center (NOC) in Santa Barbara, California, is similarly protected with strict security measures.

Expertcity's access routers are configured to watch for denial of service (DoS) attacks and log-denied connections. Multi-layer perimeter security is provided by a pair of firewalls: one between the Internet and web servers, another between the GoToMyPC broker and back-end databases. The security of this architecture has been independently confirmed by penetration tests and vulnerability assessments conducted by TruSecure Corporation[4]. Expertcity has achieved TruSecure SiteSecure Certification[5]. Quarterly perimeter tests ensure that Expertcity continues to meet all SiteSecure Certification requirements.

GoToMyPC is supposedly firewall-friendly. A PC loaded with the GoToMyPC software generates only *outgoing* HTTP/TCP traffic to ports 80, 443 and/or 8200. Most corporate firewalls are already configured to permit outgoing traffic, hence, no specific configuration is required to be carried out on the firewalls. Based on the same argument, GoToMyPC is compatible with remote desktops using dynamic IP addresses or NAT or PAT. I will also determine in a later section whether the traffic is legitimate HTTP, hence compatibility with application proxy firewalls.

All traffic between GoToMyPC browser client and remote PC is protected with 128-bit

AES[6] encryption. Specifically, AES in CFB[7] mode.

The GoToMyPC site also contained technical documents comparing itself to other technologies, most notably VPN and Symantec's PC Anywhere®. I summarized the comparison into the following table:

| | GoToMyPC | VPN | PC Anywhere® |
|---|---|---|---|
| **Software installation** | Only required on the PC to be accessed. At the other end, a Java-enabled web browser would do. | Software must be installed on VPN clients. | Software must be installed on the client as well as the remote PC (host). |
| **Configuration** | Self-configuring. | VPN client must be configured. | PC Anywhere® must be configured. |
| **Firewalls** | No changes required. | Requires opening of special ports like IPSEC. | Requires opening of special ports (incoming). |
| **NAT** | Compatible. | Depends on product. Some may not work well with NAT. | Unlikely to work. |
| **IP reliance** | Non-protocol specific. | IP-centric. | Non-protocol specific. |
| **Management of remote clients** | Since no software required on client PCs, just a web browser. | Sometimes, corporate policies and software updates have to be pushed down to the VPN clients. | Managing a corporate roll-out of PC Anywhere is complex and it involves license management. |

## 3. Network Set-up for Analysis

The following network was set up to capture and study the GoToMyPC network traffic. In my home network, I used a spare Windows 98 SE PC to install the GoToMyPC software. The Windows PC was loaded with Tiny Personal Firewall and Norton Anti-virus. A NAT/Firewall router dished out dynamic IP addresses to the internal PC clients. The NAT/Firewall router has a 4-port integrated 10/100Mbps Ethernet switch. The WAN interface of the router is connected to a 10/100Mbps Ethernet hub, which in turn connects to the cable modem. A Slackware Linux box (kernel 2.4.20) with 2 NICs sits with 1 NIC listening promiscuously on the external segment, the other connects to the internal segment.

The Slackware box was configured to run tcpdump at startup, writing to a binary dump file. The dump file is rotated at the end of each day. For remotely managing this box, SSH was used.

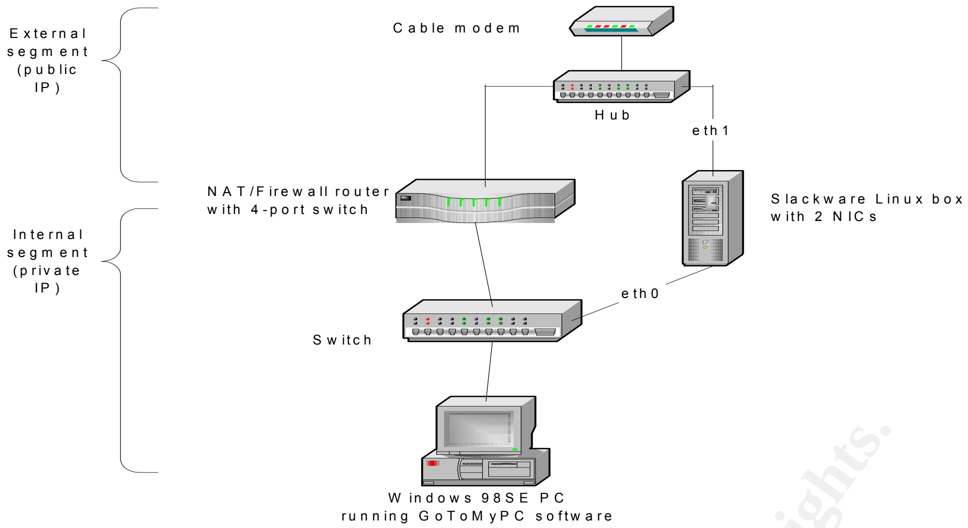



**Figure 2** Network set-up

## Installing and running the GoToMyPC software

To download the trial GoToMyPC software, I have to enter my credit card information in the online form. Upon registering for the trial, I received an email indicating the expiry of my trial and how to go about canceling the trial. Next, the GoToMyPC software was downloaded on the Windows 98 PC. 2 files: gosetup.exe and setup.exe were downloaded. Installation of the software was straightforward. During the process, I was prompted for ID and access code[8] for the PC. After which, I restarted the PC for the changes to take effect.

Upon startup, Tiny detected outgoing TCP connections to *poll.gotomypc.com* ports 80, 443 and 8200 from the executable C:\Program Files\expertcity\gotomypc\g2comm.exe (supposedly the servlet). A virus scan (Norton) on the hard disk did not reveal any suspicious Trojans or executables in the machine. Refer to Figures 3 and 4.

## Accessing the remote PC

I accessed the home PC from my office desktop using IE6.0. Logging on to my GoToMyPC account, I was presented with a list of my PCs which are online. A click of the “Connect” button brought of up a window where I have to enter the access code. Finally, my home desktop was presented to me in a window. The whole process took roughly a minute to complete. I tried out some Windows activities such as drag-and-drop, file transfer. Although the response was a bit lethargic, the action was carried out eventually. Figure 5 shows how the remote desktop is presented in a browser.

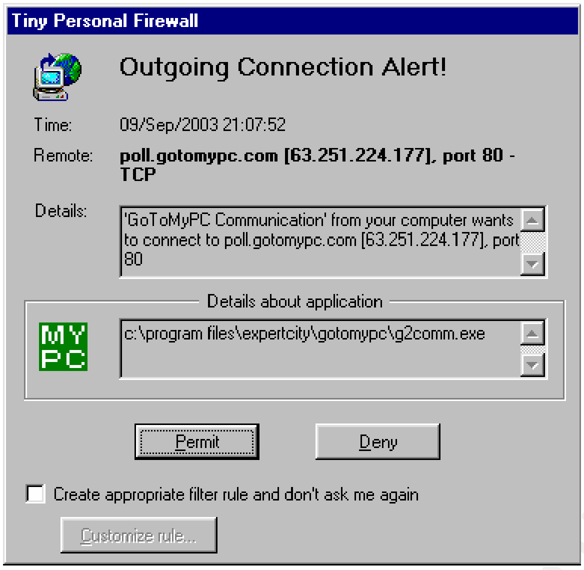



**Figure 3** Detection of outgoing connections from GoToMyPC

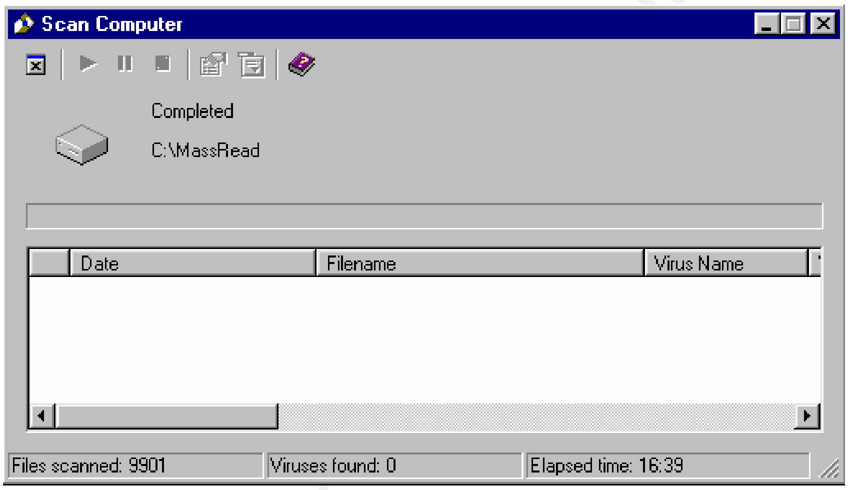



**Figure 4** Result of anti-virus scan of hard disk of PC

### Analyzing the raw dump

I analyzed a day of raw traffic dump collected on 25th Jul, with the GoToMyPC activated on the Windows 98 PC. Firstly, I ran the dump through snort with a standard rulebase:

```
$ snort -r 20030725-2359.tcp -c /usr/local/snort/snort.conf -dbl
/var/log/snort
```

Other than those known alerts that were captured off the net (e.g. MS-SQL worm, SCAN SOCKS), there were no other alerts registered. I used Ethereal to nail down to the time when the GoToMyPC was activated. The first communication packet from the PC was a SYN to 63.251.224.177 port 8200, which resolved to poll.gotomypc.com. All subsequent GoToMyPC traffic was initiated from the PC.

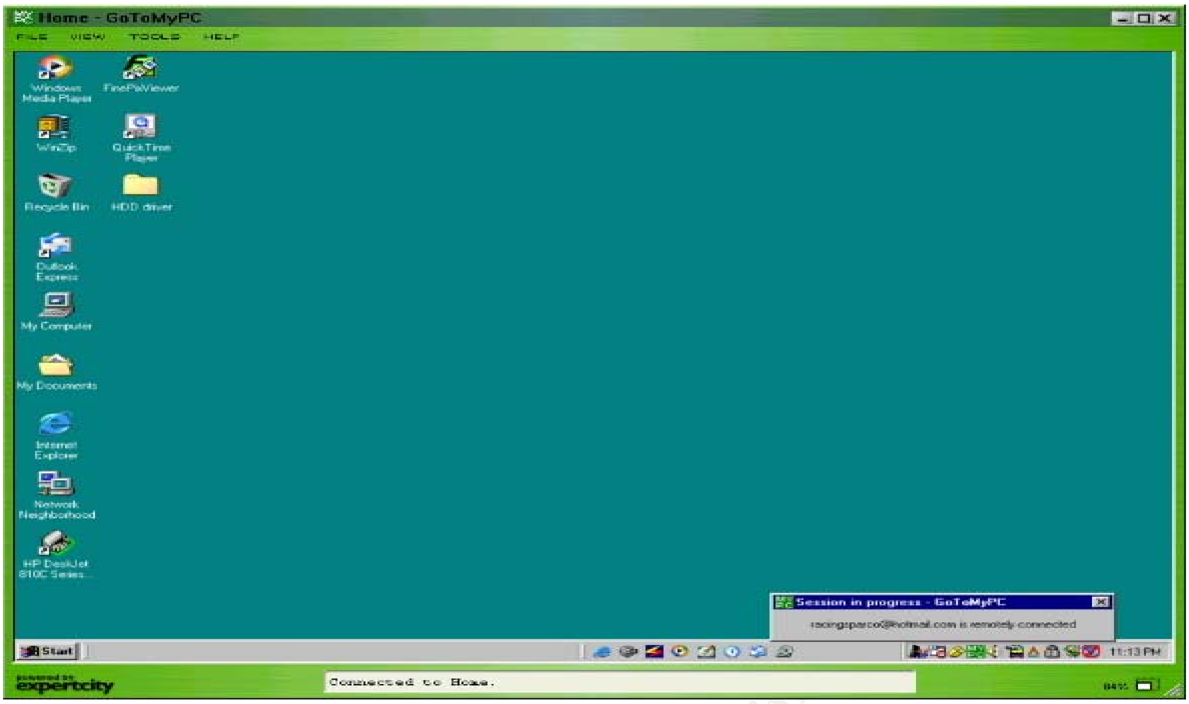



**Figure 5** My remote PC's desktop

The payloads of the TCP packets exchanged between the PC and the server suggested that HTTP version 1.0 (RFC1945[9]) based commands were used. The PC issued HTTP GET commands in the form "GET / <request> HTTP/1.0". The server replied with "HTTP/1.0 OK 200 OK <data>".

Each HTTP transaction lasted for less than a second, with a single data packet (via TCP PUSH) exchanged in the process.

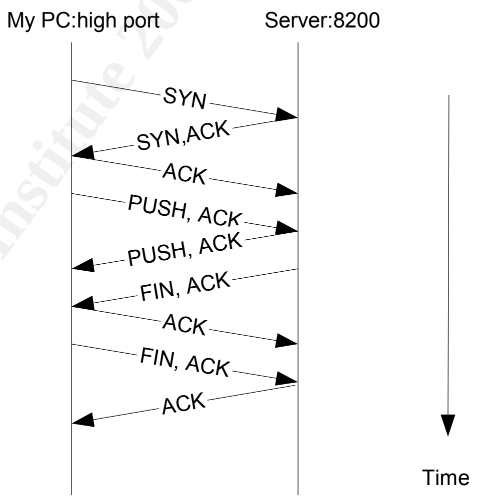



**Figure 6** HTTP transaction between PC and server

Using the "Follow TCP Stream" function of Ethereal, I observed the single-packet HTTP exchanges between the PC and poll.gotomypc.com:

| **Source**<br>**My PC** | **Destination**<br>**poll.gotomypc.com** | **Observations** |
|---|---|---|
| GET<br>/servlet/com.ec.ercbroker.servlets.PingServlet HTTP/1.0 | HTTP/1.0 200 OK<br>Pragma: no-cache<br>Content-Type: text/plain<br>Content-Length: 41<br><br>ERCBroker broker<br>http://www.gotomypc.com | First HTTP transaction. The servlet probably informing GoToMyPC of it being "live" or "up". |
| GET<br>/erc/GetOptions?build=275&platform=win32&machinekey=792680&random=**0e6db2cd25……….d%3d** HTTP/1.0 | HTTP/1.0 200 OK<br>Content-Type: text/plain<br>Content-Length: 992<br><br>0<br>random=**7df03f0e20df874b4f7097221ec7df55xtra=BQkN6B2riSme……………** | Next transaction, random keys were exchanged. I could only deduce these were part of some session key exchange mechanism. The random keys might be used to generate a session encryption key.<br><br>"build=275" might indicate the build version of the servlet. The operating platform of the PC (Win32) was also made known in the exchange. |
| GET<br>/erc/Poll?machinekey=792680&eventid=17454924&build=275&platform=win32&nc=42<br>HTTP/1.0 | HTTP/1.0 200 OK<br>Content-Type: text/plain<br>Content-Length: 67<br><br>0<br>cnt=0<br>eventid=17454924<br>purl=http://66.151.150.190/311.txt<br>pcnt=5 | The keyword "Poll" might indicate that the servlet is requesting for the address(es) of any nearby GoToMyPC communication servers. poll.gotomypc.com does not seem to be a communication server.<br><br>True enough, a Poll URL ("purl") was returned. The URL points to a text file 131.txt. |

Next, we see the HTTP exchange between the PC and the "purl" obtained earlier:

| **My PC** | **66.151.150.190** | **Observations** |
|---|---|---|
| GET /311.txt?nc=42<br>HTTP/1.0 | HTTP/1.0 200 OK<br>Content-Type: text/plain<br>Content-Length: 311<br>Pragma: no-cache<br><br>This document is used by Expertcity to probe your network connectivity<br>to our Desktopstreaming and GoToMyPC servers. These probes allow<br>us to optimize the performance of your screen sharing sessions by directing you to the best server. For additional information, please contact<br>customersupport@expertcity.com. | The servlet issues a GET for the text file 131.txt from 66.151.150.190. The content of 131.txt explained the purpose of this request. I would think that the servlet uses this HTTP GET request to determine the response or round-trip delay to the communication server (in this case, 66.151.150.190). |

Subsequent HTTP between the PC and poll.gotomypc.com were most likely keepalives:

| Source<br>My PC | Destination<br>poll.gotomypc.com | Observations |
|---|---|---|
| GET /l?792680=17454924N90 HTTP/1.0 | HTTP/1.0 200 OK<br>Content-Type: text/plain<br><br>1 | The keepalive occurred about every 15s. 17454924 probably indicated a keepalive packet. The number after "N" increments every keepalive. |

The following communication servers were observed. The servers were hosted on different sites to achieve higher availability and redundancy. Probes to these servers occurred about every 15s too. As mentioned earlier, the servlet uses these probes to determine the "best" communication server to assign for a remote access session at any point in time.

| Communication server IP | Remarks |
|---|---|
| 66.162.64.62 | Address block belonged to Time Warner Telcom |
| 66.151.115.190 | Address block belonged to Expertcity |
| 63.209.15.126 | Resolved to unknown.level3.net |
| 66.151.150.190 | Address block belonged to Expertcity |
| 64.74.80.187 | Address block belonged to Expertcity |
| 63.209.15.70 | Resolved to unknown.level3.net |

So how would the servlet know about any request for a remote access connection? Zooming in to the packets exchanged before a remote access session was started, I noted that this was communicated to the servlet via a HTTP reply packet from poll.gotomypc.com. The content "eventid=17876104" in the payload probably indicated this. The servlet then followed up with a "GET /Jedi?request…." to server 63.209.15.70, followed by a series of single-packet HTTP exchanges for 30s. While the remote access session was active, the TCP connection between the PC and server was maintained, unlike in other activity, the TCP connection only lasted one packet exchange.

Figure 7 describes the process flow observed so far.

### Assessment of the GoToMyPC Solution

Overall, the solution was quite neat. Expertcity put in a lot of effort to convince customers of their commitment to security. The security measures put in place were clearly defined and detailed in the technical documents hosted on their site, which goes to show that this is indeed a serious piece of software or solution. There is even an enterprise solution for corporate users. As Tolly noted, there is no evil intention on the part of Expertcity.

The thought of a group of corporate machines in constant contact with an external or 3rd-party service providers may not go down well with most network/security administrators or managers. However, if an organization deliberately subscribed to such a solution, the risk could be properly contained because you know who is using the service, and control the type of access rights given to the GoToMyPC client residing on the corporate desktop. Site or organizational security policies can be implemented at global, group or user levels.

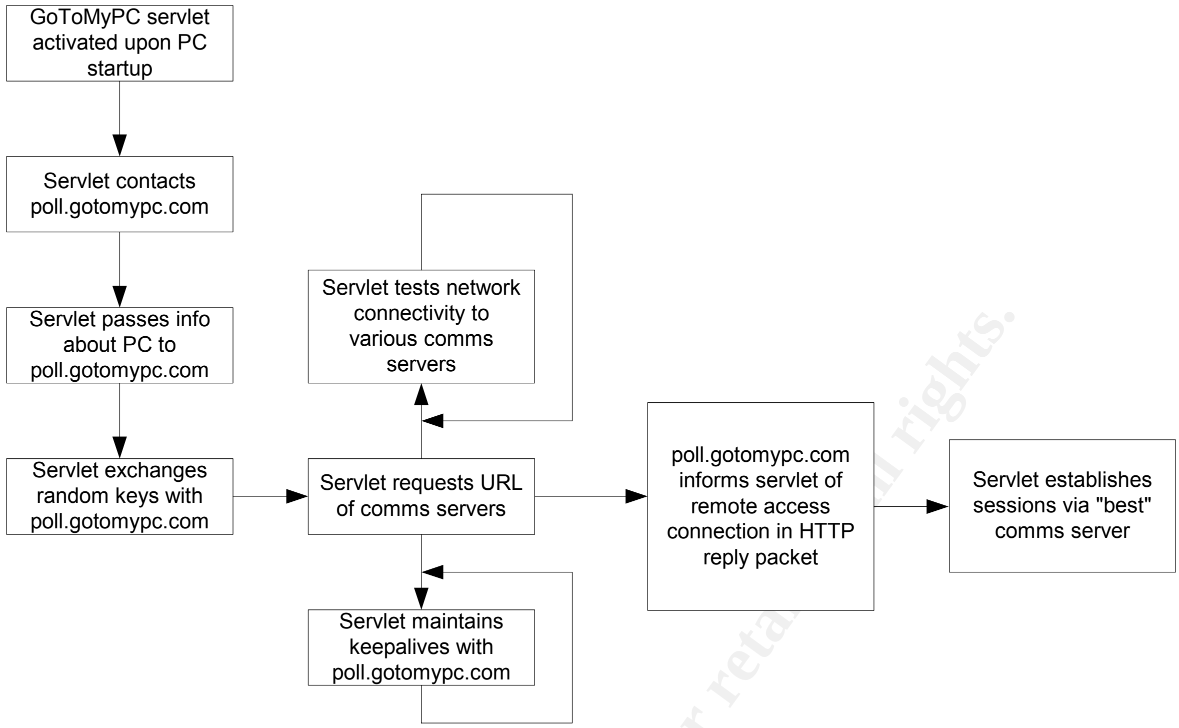



**Figure 7** Process flow upon activation of GoToMyPC servlet

The threat arises when users install and run the software without the knowledge of the organization, circumventing the security policies in the process. Once a user gained access to his corporate desktop, his access rights would be just like him being physically at his desktop. Note that access to the remote desktop can be carried out from just about anywhere, as long as a Java-enabled browser is available e.g. from home, Internet café, Network gaming centers. You can then think of all the network security breaches possible. An ignorant user might use simple passwords, or might subject to shoulder-surfing, social engineering.

Most security policies necessitate the inspection of incoming network traffic, but outgoing traffic is seldom scrutinized. To counter the threat of rogue uses of such software, I recommend the following measures:

**Inspection of outgoing traffic**

In the analysis of the raw traffic dump, the GoToMyPC activities did not trigger any alerts from a fairly standard Snort rulebase. From the observations in the previous sections, the following Snort rules could be applied to detect the presence of any GoToMyPC instances in the network:

```
alert TCP $HOME_NET any -> $EXTERNAL_NET any (msg: "Possible instance of
GoToMyPC in network"; content: "GET /erc/Poll?machinekey";)
alert TCP $HOME_NET any -> $EXTERNAL_NET any (msg: "Possible activation of
GoMyPC remote access session!!"; content: "GET /Jedi?request=";)
```

Note that we can block GoToMyPC altogether by disallowing access to poll.gotomypc.com and a list of other Expertcity/GoToMyPC IP addresses in the perimeter firewall. However, this method will not work if the software can re-configure itself to point to another location.

**Proxy server or application proxy firewall**

HTTP 1.0 commands were used in the information exchanges between the servlet and the GoToMyPC servers. I do not see a problem in the solution working behind a proxy server or firewall. Rules could be implemented to detect GoToMyPC HTTP traffic and block them. They could follow the Snort signatures in the previous section.

**Software scan on desktops**

Conduct a periodic scan of the desktops in your organization, looking for non-standard issue software installed (in the case of GoToMyPC, go2comm.exe). The desktop usage policy of your organization should be communicated to the users, warning them on the dangers of installing illegal software.

**User education**

The users would have to be informed of the possible risks of such software, even though they are legitimate. If the organization allowed the use of such software, then good security practices have to be observed, such as use of non-trivial password, screen savers or desktop disable functions when away.

**Other Remote Access Solutions**

I also tried out TotalRC version 1.20[10]. One thing I liked about this site was that in order to download the trial, you do not need to submit your credit card information. Other than that, the software allows you to set whichever outgoing port to use. However, it does not do too well in the usability department. Screen updates were not instantaneous and keyboard entries have to be sent through another Window.

There was also eBLVD Remote from ENC Technology Corporation[11]. Due to the requirement to upload my credit card information (again?) in order to download the trial, I decided not to try out this software.

## 4. Conclusion

Security is a never-ending cycle. In this paper, we looked at another set of threats arising from legitimately packaged software solutions. They boast the ability to solve deployment issues previously faced by traditional remote access solutions, such as VPN and PC Anywhere. However, these solutions make use of outgoing connections to establish screen sharing sessions, which are normally not scrutinized. To put it bluntly, the same way how Trojans communicate. These solutions have the ability to integrate in almost any existing environment with firewalls, NAT etc. The use of such software in a controlled environment is acceptable, provided good security practices are adopted by the users. The threat comes from rogue use of the software, without the knowledge of the organization. The seriousness of this threat is real. There will always be users/employees trying to circumvent the organization's security policies – either knowingly or unknowingly. Other than the security measures recommended at the network layer, user education and awareness is critical in mitigating this sort of risks.

# Part 2: Network detects

## 1. Detect #1: Scan Squid and Proxy (8080) attempts

### Source of trace

The rawdump file used for this detect was 2002.4.31 and obtained from http://www.incidents.org/Raw/logs.

Looking at the rawdump, the OUI of the source and destination MAC addresses (00:03:E3 and 00:00:0C) belonged to CISCO Systems. Hence, I suspect the IDS probe was placed in between 2 CISCO devices. A quick glance at the IP addresses revealed that the scanned network was a class B (226.185.X.X).

The incorrect checksums reported were ignored due to the fact that the IP adresses have been tampered with ( http://www.incidents.org/Raw/logs/README ).

### Detect was generated by

I used Snort version 1.9.1 (Build 231) and the *rules* file dated 13 May 2003 (with a default snort.conf). The command run was:

```
# snort -dr 2002.4.31 -c ~snort/var/rules/snort.conf -bl ~snort/var/log &
```

2 files were subsequently created in ~snort/var/log:

```
# ls -l ~snort/var/log/
total 81416
-rw-------   1 root     root     67170142 May 13 15:43 alert
-rw-------   1 root     root     16104188 May 13 15:43 snort.log.1052869387
```

A sample of the alert file generated was:

```
[**] [1:620:2] SCAN Proxy (8080) attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/30-16:02:57.834488 216.13.66.30:3841 -> 226.185.141.57:8080
TCP TTL:113 TOS:0x0 ID:58481 IpLen:20 DgmLen:48 DF
******S* Seq: 0xF7C9D762  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] [1:620:2] SCAN Proxy (8080) attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/30-16:02:57.834488 216.13.66.30:3839 -> 226.185.141.56:8080
TCP TTL:113 TOS:0x0 ID:58479 IpLen:20 DgmLen:48 DF
******S* Seq: 0xF7C8555E  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] [1:618:2] SCAN Squid Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/30-16:02:57.834488 216.13.66.30:3842 -> 226.185.141.57:3128
TCP TTL:113 TOS:0x0 ID:58482 IpLen:20 DgmLen:48 DF
******S* Seq: 0xF7CA97E4  Ack: 0x0  Win: 0x4000  TcpLen: 28
```

```
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] [1:620:2] SCAN Proxy (8080) attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/30-16:02:57.834488 216.13.66.30:3843 -> 226.185.141.58:8080
TCP TTL:113 TOS:0x0 ID:58483 IpLen:20 DgmLen:48 DF
******S* Seq: 0xF7CB2602  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] [1:618:2] SCAN Squid Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/30-16:02:57.834488 216.13.66.30:3840 -> 226.185.141.56:3128
TCP TTL:113 TOS:0x0 ID:58480 IpLen:20 DgmLen:48 DF
******S* Seq: 0xF7C93C93  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
```

A whole list of SCAN Proxy (8080) and Squid Proxy attempts were triggered. Running a combination of grep, uniq and sort, I was able to generate a statistical listing of the alerts generated:

```
# cat alert | grep '\[\*\*\]' | sort | uniq -c | sort -rn | cat >
alert.stats &
# cat alert.stats
 103873 [**] [1:618:2] SCAN Squid Proxy attempt [**]
 102496 [**] [1:620:2] SCAN Proxy (8080) attempt [**]
     57 [**] [1:1616:4] DNS named version attempt [**]
     10 [**] [1:628:1] SCAN nmap TCP [**]
      1 [**] [1:498:3] ATTACK RESPONSES id check returned root [**]
      1 [**] [116:45:1] (snort_decoder) TCP packet len is smaller than 20
bytes! [**]
```

Due to the high frequency of Proxy scans, I shall based my analysis on them. The triggering rule for the SCAN Squid Proxy and Proxy attempts was:

```
# cat scan.rules | awk '/8080/ || /3128/ {print $0}'

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;)
```

These Snort signatures look for any TCP SYN packets to destination ports 3128 and 8080.

## Probability the source address was spoofed

I sieved out the SYN SCAN packets into another binary file for ease of analysis (with the help of text-Ethereal) :

```
# tethereal -r 2002.4.31 'tcp.flags.syn==1 and tcp.flags.ack==0' -w
syn.only.bin

# tethereal -r syn.only.bin | awk '{print $4}' | sort | uniq -c | sort -rn
> syn.source.stats

# cat syn.source.stats
 206363 216.13.66.30
      6 194.108.153.205
```

That's a lot of SYN packets coming from 216.13.66.30, over a period of about 1.5

hours. A check with whois revealed the source of both IP addresses:

OrgName: ATT Canada Telecom Services Company
NetRange: 216.13.0.0 to 216.13.255.255
CIDR: 216.13.0.0/16

OrgName: TIPI (Netherlands)
NetRange: 195.108.153.0 - 195.108.153.255
CIDR: 195.108.153.0/24

I did not suspect the IP addresses were spoofed because being SYN packets, the attacker required replies from the target in order to complete the connection.

## Description of the attack

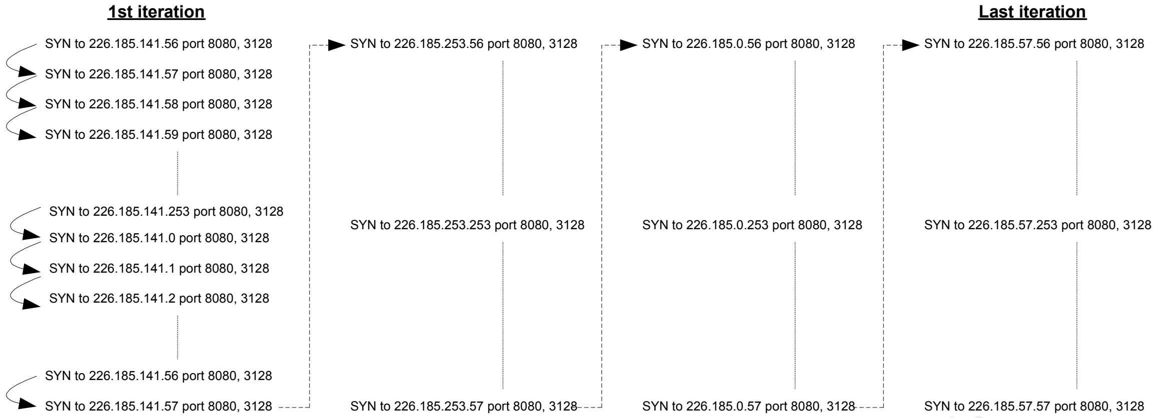
The attack involved a reconnaissance attempt targeted at the 226.185.0.0/16 block of addresses, probing for hosts listening on TCP ports 8080 or 1328. Squid proxies are usually configured to listen on tcp/3128. The attacker might be looking for vulnerable proxy servers (e.g. Squid or WinGate) or for open proxies. A SYN-ACK response would indicate the presence of such services. There are numerous vulnerabilities associated with mis-configured Squid and WinGate proxy servers.

If the attacker was looking for open proxies, then it did not matter which type of proxy server was running. The attacker could then use the open proxy for spamming[12] and even DOS attacks against IRC servers.

## Attack mechanism

The first SYN packet from the attacker arrived at 1602 hrs on 30th May 2002, and the last at 1931 hrs on the same day. Within a timespan of about 1.5 hours, a total of 206,363 SYN packets were detected and 43433 target IP addresses were scanned.

There was a pattern in the way the SYN SCAN packets were generated. Each SYN SCAN cycle starts from IP address A.B.C.56 to A.B.C.253, resets, and continued from A.B.C.0 to A.B.C.57. The Class C block of the next cycle is determined by incrementing the 3rd octet by 1. When the 3rd octet reaches 253, the next cycle will begin from the third octet equal to zero (0) and so on. Each cycle consisted of 255 IP addresses and the time taken for SYN SCAN each cycle was approximately a minute. The SYN SCAN ends when the 3rd octet equals to 57. A simple diagram illustrating the SYN SCAN pattern:




The following observations were made of the SYN packets originating from 216.13.66.30:

– Source port used in the range 1026 to 5000
– Source port increments by 1 every SYN
– IP ID increments by 1 every SYN – this is the behavior of the TCP/IP stack of a number of OSes e.g. FreeBSD, Solaris 7, AIX 4.3, Win 2000, just to name a few
– Same IP TTL value of 113 for all packets (likely the initial TTL was 128, pointing to a Windows NT/2000 machine)
– Window size of 16384
– TCP/IP options: MSS-1460 NOP NOP SACK Permitted

To determine the OS of the attacker, p0f was used with the following responses:

```
# p0f -s snort.log.1052868990 | more
p0f: passive os fingerprinting utility, version 1.8.3
(C) Michal Zalewski <lcamtuf@gis.net>, William Stearns <wstearns@pobox.com>
p0f: file: '/etc/p0f.fp', 207 fprints, iface: 'wlan0', rule: 'all'.
216.13.66.30 [16 hops]: Windows 2000 (9)
```

An automated tool was probably involved, based on:

– the pattern of target IP addresses,
– high rate of scans, roughly 2300 SYN packets per minute, and
– incrementing of source port by 1 every SYN

# Correlations

In Don Murdoch's posting in *incidents.org*[13] dated 21 Apr 2003 (on rawdump 2002.4.30), he discussed similar sightings of SCAN Proxy and Squid Proxy attempts but these were directed at a particular IP address (226.185.177.57).

A search of Dshield's mailing list archive during the period of Apr to May 2002 showed an instance of large amounts of scans on ports 8080, 3128 and 80 detected. The thread[14] described the payload that was used to test whether the proxy is open. I checked whether there were any replies in the 226.185.0.0/16 network:

```
# tethereal -r 2002.4.31 -n 'ip.dst==216.13.66.30 and tcp.flags.syn==1 and
tcp.flags.ack==1'
```

Unfortunately, there were none, hence I was not able to look at the payload should a

3-way handshake succeed. I also could not find any instances of the IP address within +/- 10 days of 2002.4.31.

## Evidence of active targeting

The scanning pattern described earlier suggests an automated tool was involved to generate the SYN packets. The barrage of SYN packets were fired to locate servers with open TCP ports 8080 and 3128 in the Class B network of 226.185.0.0/16. Hence, there was evidence of active targeting of the network, but not at any particular host.

## Severity

| ***Severity =*** | ***Value*** | ***Remarks*** |
|---|---|---|
| *(Criticality* | 4 | The scan activity was part of a reconnaissance attempt to locate any listening proxies in 226.185.0.0/16. If the attacker solicited a response from an active proxy, he could either use it for spam activity, or as a springboard to attack other sites. |
| + | | |
| *Lethality)* | 2 | The scans would not classify as lethal, because they were basically reconnaissance probes. However, the knowledge of open proxies within the network may be lethal as explained earlier. |
| - | | |
| *(System countermeasures* | 4 | There were no SYN-replies to the attacking IP. 3 possibilities:<br>– no proxy servers present<br>– proxy server located in the internal network and behind a firewall, the latter discarding the SYNs silently<br>– access list implemented in proxy server, accepting requests from the internal network only<br>Assuming the third possibility, a high score was given. |
| + | | |
| *Network countermeasures)* | 1 | The fact that the SYN packets were captured by the IDS indicated that they at least passed the perimeter router. The only network countermeasure in place would be the IDS probe, which monitored incoming/outgoing traffic of 226.185.0.0/16. |
| ***Result =*** | **1** | Low severity score. |

## Defensive recommendations

If there was a proxy server in the network, proper access list should be in place to prevent abuse. Such as allowing only internal IP to access the proxy services.

If there were no proxy servers in the network, then such reconnaissance attempts should be blocked at the border router or firewall. This would in turn reduce the amount of IDS logs.

For example,

```
access-list 101 deny tcp any 226.185.0.0 0.0.255.255 eq 3128
access-list 101 deny tcp any 226.185.0.0 0.0.255.255 eq 8080
```

## Multiple choice question

Given a raw binary dump file, what are the snort options to sieve out only SYN packets into another binary file called syn.bin?

a. -r *dumpfile* -v 'tcp[13] = 0x2' -bl *logdir*
b. -r *dumpfile* 'tcp[13] = 0x2' -l *logdir*
c. -r *dumpfile* 'tcp[12:2] & 0xfff0 = 0x2' -bl *logdir*
d. -r *dumpfile* 'tcp.flags.syn==1 and tcp.flags.ack==0' -bl *logdir*

The answer is (a). Answer (b) logs in ASCII. Answer (c) semantically incorrect, no output will be generated. Answer (d) is syntactically incorrect, because the filter is Ethereal-specific.

*Result of post to intrusions@incidents.org*

This detect was posted to intrusions@incidents.org on 10 Jun 2003. One reply was received with 5 questions and 2 comments, which were noted and rectified.

```
*From:* "Brian Coyle" <brian@linuxwidows.com>
*To:* "Johnny Wong (Singapore)" <deepcrack2002@yahoo.com>,
intrusions@incidents.org
*Subject:* Re: LOGS: GIAC GCIA Version 3.3 Practical Detect
*Date:* Wed, 11 Jun 2003 00:35:19 -0400
*CC:* johnny_wong@ida.gov.sg

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

On Tuesday 10 June 2003 23:40, Johnny Wong \(Singapore\) wrote:

> Detect #1: Scan Squid and Proxy (8080) attempts

[massive snippage thru-out]

> [root@goober 1]# cat alert | grep '\[\*\*\]' | sort |
> uniq -c | sort -rn | cat > alert.stats &
> [root@goober 1]# cat alert.stats
>  103873 [**] [1:618:2] SCAN Squid Proxy attempt [**]
>  102496 [**] [1:620:2] SCAN Proxy (8080) attempt [**]
>      57 [**] [1:1616:4] DNS named version attempt [**]
>      10 [**] [1:628:1] SCAN nmap TCP [**]
>       1 [**] [1:498:3] ATTACK RESPONSES id check
> returned root [**]
>       1 [**] [116:45:1] (snort_decoder) TCP packet len
> is smaller than 20 bytes! [**]
>

Nice that you're showing your work.

> [root@goober rules]# tethereal -r syn.only.bin | awk
> '{print $4}' | sort | uniq -c | sort -rn >
> syn.source.stats
>
> [root@goober 1]# cat syn.source.stats
>  206363 216.13.66.30
>       6 194.108.153.205
>
> That's a lot of SYN packets coming from 216.13.66.30.
```

```
A lot?  Over what period of time?   1 day? 1 year?
You quantify this later in section 5, but you probably
should mention it here too.

>
> 4.Description of the attack
>
> The attack involved a reconnaissance attempt targeted
> at the 226.185.0.0/16 block of addresses

The whole block?   The sample you showed only had 1 target.
You don't discuss add'l targets until later... What kind of
script-fu would you use to to determine/summarize the targets?

> The SYN packets do not look to be crafted because the
> source port incremented sequentially from 1026 to
> 5000. The IP ID too incremented sequentially. The IP
> TTL value was consistent at 113.

What is the significance of the IP ID and TTL values?
What clues does this offer?   Can any passive fingerprinting
be done on the attacker (don't forget the TCP/IP options)?

> An automated tool was probably involved.

Any guesses as to which tool?   Any clues in how the
address range was scanned?

> As there were no listeners on ports
> 8080 and 3128 in the 226.185.0.0/16 network

How do you know this?   Would a snort rule cause an alert
to be logged if there was a reply?

> There were no servers listening to TCP ports
> 8080 and 3128. I assumed that if there were, then ?no
> replies? to unsolicited SYNs (firewall?) would
> indicate that access controls were in place.

Are you sure of this after you answer the question above?

>
> 10.Multiple choice question
>
> Given a raw binary dump file, what is the snort
> command

Given what you list below, shouldn't this be 'what
snort OPTIONS...'?

> to sieve out only SYN packets into another
> binary file called syn.bin?
>
> a.  -r dumpfile -v 'tcp[13] = 0x2' -bl logdir
> b.  -r dumpfile 'tcp[13] = 0x2' -l logdir
> c.  -r dumpfile 'tcp[12:2] & 0xfff0 = 0x2' -bl logdir
> d.  -r dumpfile 'tcp.flags.syn==1 and tcp.flags.ack==0'
> -bl logdir
>
> The answer is (a).

But how does the file syn.bin get created?

- --
Linux - the ultimate Windows Service Pack
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.2.1 (GNU/Linux)
```

```
Comment: Brian Coyle, GCIA
http://www.giac.org/GCIA.php
```


iD8DBQE+5rGVER3MuHUncBsRAvtSAJ9XEy3PhVBQkfriv4dZa/ia/p2sNgCfaQvn
6EccjViTFavJHEaojFvm85Y=
=r2w5
-----END PGP SIGNATURE-----


# 2. Detect #2: ACK scan attempts

## Source of trace

The rawdump file used for this detect was 2002.6.9 and obtained from http://www.incidents.org/Raw/logs.

Looking at the rawdump, the OUI of the source and destination MAC addresses (00:03:E3 and 00:00:0C) belonged to CISCO Systems (reference to http://standards.ieee.org/regauth/oui/oui.txt). Hence, I suspected the IDS probe was placed in between 2 CISCO devices.

The incorrect checksums reported were ignored due to the fact that the IP adresses have been tampered with (http://www.incidents.org/Raw/logs/README).

## Detect was generated by

I used to Snort version 1.9.1 (Build 231) and rules file dated 13 May 2003 (with a default snort.conf). The command run was:

```
# snort -dr 2002.6.9 -c ~snort/var/rules/snort.conf -bl ~snort/var/log &
```

Snort reported a whole list of SCAN nmap TCP attempts. Running a combination of grep, uniq and sort, I generated a statistical listing of the alerts generated:

```
# cat alert | grep '\[\*\*\]' | sort | uniq -c | sort -rn | cat >
alert.stats &
# cat alert.stats
     81 [**] [1:628:1] SCAN nmap TCP [**]
     45 [**] [1:1616:4] DNS named version attempt [**]
     27 [**] [1:1322:4] BAD TRAFFIC bad frag bits [**]
     20 [**] [1:615:3] SCAN SOCKS Proxy attempt [**]
     16 [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic [**]
     10 [**] [1:624:1] SCAN SYN FIN [**]
      4 [**] [1:621:1] SCAN FIN [**]
      4 [**] [1:620:2] SCAN Proxy (8080) attempt [**]
      4 [**] [1:618:2] SCAN Squid Proxy attempt [**]
      3 [**] [1:523:3] BAD TRAFFIC ip reserved bit set [**]
      2 [**] [116:46:1] (snort_decoder) WARNING: TCP Data Offset is less
than 5! [**]
```

A sample of the alert file:

```
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/08-08:24:26.964488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
202.29.28.1:80 -> 46.5.137.172:80 TCP TTL:46 TOS:0x0 ID:28838 IpLen:20
DgmLen:40
***A**** Seq: 0x2B0  Ack: 0x0  Win: 0x578  TcpLen: 20
```

```
[Xref => arachnids 28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/08-08:24:31.954488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
202.29.28.1:80  ->  46.5.137.172:80  TCP  TTL:46  TOS:0x0  ID:29078  IpLen:20
DgmLen:40
***A**** Seq: 0x316  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => arachnids 28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/08-08:31:12.944488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
159.226.208.40:80  ->  46.5.15.174:80  TCP  TTL:48  TOS:0x0  ID:2100  IpLen:20
DgmLen:40
***A**** Seq: 0x4A  Ack: 0x0  Win: 0x400  TcpLen: 20
[Xref => arachnids 28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/08-08:31:13.924488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
159.226.208.40:80  ->  46.5.15.174:80  TCP  TTL:48  TOS:0x0  ID:2544  IpLen:20
DgmLen:40
***A**** Seq: 0xA4  Ack: 0x0  Win: 0x400  TcpLen: 20
[Xref => arachnids 28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/08-08:31:15.334488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
211.152.3.40:80  ->  46.5.15.174:80  TCP  TTL:39  TOS:0x0  ID:3056  IpLen:20
DgmLen:40
***A**** Seq: 0x109  Ack: 0x0  Win: 0x400  TcpLen: 20
[Xref => arachnids 28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/08-08:31:16.304488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
211.152.3.40:80  ->  46.5.15.174:80  TCP  TTL:39  TOS:0x0  ID:3502  IpLen:20
DgmLen:40
***A**** Seq: 0x15E  Ack: 0x0  Win: 0x400  TcpLen: 20
[Xref => arachnids 28]
```

The triggering rule for the SCAN nmap TCP attempts was found to be:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap
TCP";flags:A;ack:0; reference:arachnids,28; classtype:attempted-recon;
sid:628; rev:1;)
```

Anytime a TCP packet with only the ACK flag set, and the ACK number equal 0, an alert would be triggered. TCP scans carried out by older versions of nmap have the ACK number set to 0, hence snort flagged such occurrences as an nmap TCP scan. Newer versions of nmap uses random non-zero ACK numbers.

## Probability the source address was spoofed

If indeed these were nmap TCP scans as reported by snort, then the source addresses would not be spoofed because the attacker needs to see the response from the target (a RST packet if the scanned port was unfiltered) as part of the information gathering attempt.

# Description of the attack

Unsolicited TCP packets with the ACK flag set and ACK number equal 0 were sent from multiple source addresses (26 of them), starting from 8th Jul 2002 0824hrs GMT. The ACK packets were destined to the one or more destination addresses within the 46.5.0.0/16 Class B network. In particular, I noticed the stream of packets targeted at address 46.5.80.149:

```
200 2002-07-09 23:28:21.424488 66.125.147.222 -> 46.5.80.149  TCP 45147 >
6346 [ACK] Seq=713 Ack=0 Win=1024 Len=0
201 2002-07-09 23:28:26.424488 66.125.147.222 -> 46.5.80.149  TCP 45147 >
6346 [ACK] Seq=783 Ack=0 Win=1024 Len=0
202 2002-07-09 23:28:31.434488  12.99.244.2 -> 46.5.80.149  TCP 80 > 6346
[ACK] Seq=843 Ack=0 Win=1024 Len=0
203 2002-07-09 23:28:36.434488  12.99.244.2 -> 46.5.80.149  TCP 80 > 6346
[ACK] Seq=903 Ack=0 Win=1024 Len=0
204 2002-07-09 23:28:41.444488   64.3.83.34 -> 46.5.80.149  TCP 80 > 6346
[ACK] Seq=979 Ack=0 Win=1024 Len=0
205 2002-07-09 23:28:46.434488   64.3.83.34 -> 46.5.80.149  TCP 80 > 6346
[ACK] Seq=22 Ack=0 Win=1024 Len=0
206 2002-07-09 23:28:51.474488  65.113.31.2 -> 46.5.80.149  TCP 80 > 6346
[ACK] Seq=110 Ack=0 Win=1024 Len=0
207 2002-07-09 23:28:56.444488  65.113.31.2 -> 46.5.80.149  TCP 80 > 6346
[ACK] Seq=188 Ack=0 Win=1024 Len=0
208  2002-07-09  23:29:01.444488 206.111.234.194  ->  46.5.80.149   TCP  80  >
6346 [ACK] Seq=280 Ack=0 Win=1024 Len=0
209  2002-07-09  23:29:06.444488 206.111.234.194  ->  46.5.80.149   TCP  80  >
6346 [ACK] Seq=374 Ack=0 Win=1024 Len=0
```

I observed a pattern in the sequence of packets:
- ACK packet sent to 46.5.80.149 port 6346 at 5s intervals
- 2 ACK packets sent from each source IP
- low TCP sequence number (below 1000)
- low source port used in most of the packets (i.e. port 80)
- Window sizes of 1024 and 1400
- TTL value from 45 to 47

The same exact pattern was observed in consecutive rawdumps on 2002.6.10 to 2002.6.11 and on 2002.6.15 at the following times:

| Date | Time (GMT) | No. of Source addresses |
|---|---|---|
| **2002-07-10** | **04:03** | **4** |
| **2002-07-10** | **09:26** | **2** |
| **2002-07-11** | **01:12** | **5 **repeated source addresses** |
| **2002-07-11** | **18:44** | **7 **repeated source addresses** |
| **2002-07-11** | **19:39** | **5 **repeated source addresses** |
| **2002-07-11** | **22:27** | **5 **repeated source addresses** |
| **2002-07-11** | **23:26** | **5 **repeated source addresses** |
| **2002-07-15** | **16:47** | **2** |
| **2002-07-15** | **18:11** | **5 **repeated source addresses** |
| **2002-07-15** | **19:24** | **5 **repeated source addresses** |

**2002-07-15 20:52 5 **repeated source addresses**
**2002-07-15 23:11 5 **repeated source addresses**

The logs did not show any responses from 46.5.80.149. Could this be an active targeting of the IP? The Gnutella destination port in these ACK packets made me look further into this particular "attack".

## Attack mechanism

As extracted from *man nmap*, ACK scans are used in reconnaissance attempts to map out the firewall rulesets. They could also determine whether the firewall is a stateful or just a simple packet filter that blocks incoming SYN. By sending an ACK-only packet to a specified port, a returned RST packet would indicate a non-stateful packet filter. Otherwise, no response would be given.

From the earlier observations, I found it difficult to pinpoint the ACK scans to *nmap* because it would require strict coordination to send each ACK packet to the target address every 5s from different source addresses. If we argue that *nmap* could have been run from the same machine using decoy scan option, then how do we explain (i) the use of different TCP sequence number for each ACK packet, (ii) differing TTL values, (iii) ACK number of 0 considering that this peculiarity only found in older versions of *nmap* (pre-version 2.3 BETA 8) and (iv) except for packets from 66.125.147.222 which used random source ports, the others used the same source port of 80.

I also noted that from 2002-7-10 to 2002-7-12 and from 2002-7-15 to 2002-7-16, there were a lot of one-sided Gnutella CONNECTs to the this IP address from sources located in the Class B address of 148.63.0.0, 148.64.0.0 and 148.65.0.0 (all belonging to StarBand Communications). These packets were particularly TCP with only the PUSH flag set and ACK number 0.

```
16:46:19.754488      148.63.134.33.2302      >      46.5.80.149.6346:      P
843355658:843355828(170) win 8192 (DF)
0x0000   4500 00d2 d1df 4000 6f06 a850 943f 8621        E.....@.o..P.?.!
0x0010   2e05 5095 08fe 18ca 3244 960a 0000 0000        ..P.....2D......
0x0020   5e08 2000 8039 0000 474e 5554 454c 4c41        ^....9..GNUTELLA
0x0030   2043 4f4e 4e45 4354 2f30 2e36 0d0a 5573        .CONNECT/0.6..Us
0x0040   6572 2d41 6765 6e74 3a20 4265 6172 5368        er-Agent:.BearSh
0x0050   6172 6520 322e 362e 320d 0a4d 6163 6869        are.2.6.2..Machi
0x0060   6e65 3a20 312c 382c 3338 332c 312c 3339        ne:.1,8,383,1,39
0x0070   380d 0a50 6f6e 672d 4361 6368 696e 673a        8..Pong-Caching:
0x0080   2030 2e31 0d0a 486f 7073 2d46 6c6f 773a        .0.1..Hops-Flow:
0x0090   2031 2e30 0d0a 4c69 7374 656e 2d49 503a        .1.0..Listen-IP:
0x00a0   2031 3438 2e36 332e 3133 342e 3333 3a36        .148.63.134.33:6
0x00b0   3334 360d 0a52 656d 6f74 652d 4950 3a20        346..Remote-IP:.
0x00c0   3137 302e 3132 392e 3230 342e 3139 0d0a        170.129.204.19..
0x00d0   0d0a                                           ..

17:33:06.934488      148.63.153.23.3506      >      46.5.80.149.6346:      P
168117770:168117858(88) win 8192 (DF)
0x0000   4500 0080 1365 4000 6f06 5427 943f 9917        E....e@.o.T'.?..
0x0010   2e05 5095 0db2 18ca 0a05 460a 0000 0000        ..P.......F.....
0x0020   5e08 2000 98d2 0000 474e 5554 454c 4c41        ^.......GNUTELLA
```

```
0x0030   2043 4f4e 4e45 4354 2f30 2e36 0d0a 5573        .CONNECT/0.6..Us
0x0040   6572 2d41 6765 6e74 3a20 4265 6172 5368        er-Agent:.BearSh
0x0050   6172 6520 322e 342e 320d 0a50 6f6e 672d        are.2.4.2..Pong-
0x0060   4361 6368 696e 673a 2030 2e31 0d0a 486f        Caching:.0.1..Ho
0x0070   7073 2d46 6c6f 773a 2031 2e30 0d0a 0d0a        ps-Flow:.1.0....

17:34:10.924488      148.63.153.23.3506      >      46.5.80.149.6346:      P
168117770:168117858(88) win 8192 (DF)
0x0000   4500 0080 2203 4000 6f06 4589 943f 9917        E...".@.o.E..?..
0x0010   2e05 5095 0db2 18ca 0a05 460a 0000 0000        ..P.......F.....
0x0020   5e08 2000 98d2 0000 474e 5554 454c 4c41        ^.......GNUTELLA
0x0030   2043 4f4e 4e45 4354 2f30 2e36 0d0a 5573        .CONNECT/0.6..Us
0x0040   6572 2d41 6765 6e74 3a20 4265 6172 5368        er-Agent:.BearSh
0x0050   6172 6520 322e 342e 320d 0a50 6f6e 672d        are.2.4.2..Pong-
0x0060   4361 6368 696e 673a 2030 2e31 0d0a 486f        Caching:.0.1..Ho
0x0070   7073 2d46 6c6f 773a 2031 2e30 0d0a 0d0a        ps-Flow:.1.0....
```

The “Gnutella CONNECT/0.6” suggested the version of the Gnutella protocol and the string after “user-agent:” indicated the client (i.e. Bearshare) used. A Gnutella servent attaches to a network via connection to another servent. Servents obtain IP addresses of other servents from their *host cache*.

So how do the ACK scans relate to the Gnutella CONNECTs? I suspected the ACK packets were “keep-alives” between other servents and 46.5.80.149, which happened to be found in their host caches. The low source port in the ACK packets, and particularly port 80, might be used to bypass firewalls. A packet filtering firewall would allow these packets to pass, thinking they were part of an established HTTP connection.

In between these ACK packets, other servents tried to join the Gnutella network via this IP. As 46.5.80.149 was not featured in the raw dumps from 2002.6.15 onwards, I could only deduce that the respective caches timed-out on this particular IP, and the keep-alives stopped. This IP could have previously belonged to a Gnutella client, hence the reason why it ended up in the host cache in the first place[15], a common scenario in dynamic IP environments like DSL, cable Internet access.

## Correlations

I did a *whois* on the source addresses of the ACK packets and found that the packets originated from ISPs in the US. I tried to check whether any of the source has been reported in http://www.dshield.org/ipinfo.php, but to no avail.

Previous analysis of random ACK scans attributed the cause to load balancing devices, notably in:

http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00027.html[16]
http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00039.html[17]
http://cert.uni-stuttgart.de/archive/intrusions/2002/12/msg00167.html[18]

However, in this case, the difference was that the destination port was Gnutella-6346 and more than one sending host was detected.

# Evidence of active targeting

I noted that this IP did not feature in rawlogs prior to 2002.6.9. The relationship between the ACK scans and Gnutella CONNECT events suggested that the IP 46.5.80.149 was unwittingly targeted due to the fact it was previously used by a Gnutella client.

# Severity

Criticality = 1
There was no evidence of answers from 46.5.80.149 to the ACKs or Gnutella CONNECTs. A real Gnutella client would reply with something like "GNUTELLA/0.6 200 OK". The existence of a Gnutella client would not count as critical to that of a Web or DNS server.

Lethality = 1
Lethality was low. The ACKs were probably used to maintain keepalives between Gnutella servents.

System countermeasures = 4
There was a possibility that a machine existed on IP address 46.5.80.149. The observations suggested that this machine, if it existed, took over an IP address that previously belonged to a Gnutella client. Or, the client software was uninstalled. I assumed the latter and gave a high score because of the possible risks of such software.

Network countermeasures = 1
The fact that the ACK and PUSH packets were captured by the IDS indicated that they at least passed the perimeter router. The only network countermeasure in place would be the IDS probe.

Severity = (1+1) – (1+1) = 0

# Defensive recommendations

If the IDS probe was placed behind a firewall, and yet the ACK scans were detected, then a stateful firewall would do the trick in dropping these packets. Similarly, access to port 6346 from outside should be blocked, if peer-to-peer software is not allowed in the network.

# Multiple choice question

What could be the most likely reason when a network starts receiving unsolicited Gnutella CONNECT packets destined for an IP address within the network?

a. someone is performing a scan for hosts listening on port 6346
b. the IP address was previously used by a Gnutella client
c. these packets were responses to an earlier Gnutella REQUEST packets

d. a mis-configured Gnutella client

The answer is (b).

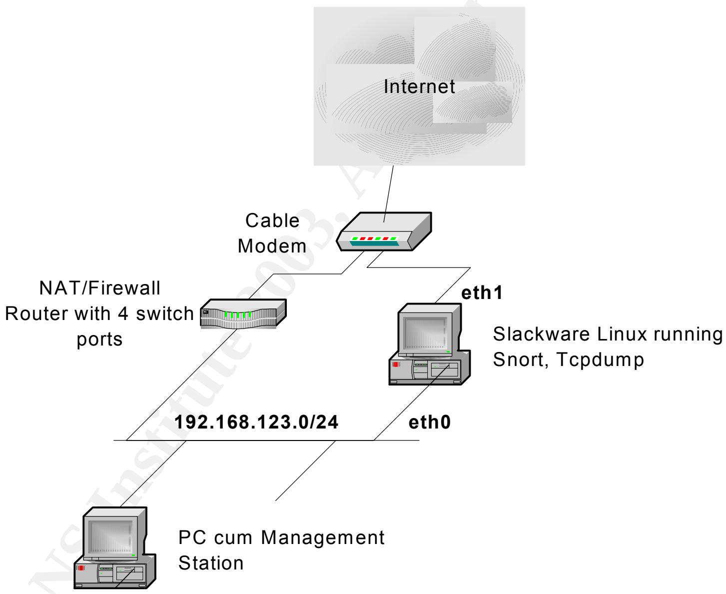
# 3. Detect #3: MS-SQL Worm Propagation Attempt

## Source of trace

The last network detect came from my home network. My home network configuration consisted of:

- a Slackware Linux box running on kernel 2.4.20 with two (2) network interfaces; one listening promiscuously for packets ingressing and egressing from my network, and the other connected to the internal LAN
- *tcpdump* was used to log the packets into a binary dump file, which got rotated every day
- the NAT/Firewall router was configured to block any traffic originating from the Internet into the internal LAN

The following diagram describes the network set-up:




The binary dumps collected from 14 to 19 Jul 2003 were analysed for this exercise.

## Detect was generated by

I used Snort version 2.0.0 (Build 72) and rules file dated 6 Mar 2003 (with a default snort.conf). The command run was:

```
# snort -r 20030714-2359.tcp -c snort.conf -dbl foo -L snort0714
# snort -r 20030715-2359.tcp -c snort.conf -dbl foo -L snort0715
.
.
.
```

```
# snort -r 20030719-2359.tcp -c snort.conf -dbl foo -L snort0719
```

Snort reported a number of MS-SQL Worm Propagation attempts against my ISP-assigned public IP address (masked out as X.X.X.X). Running a combination of grep, uniq and sort on the concatenated *alert* file, I generated a statistical listing of the alerts generated:

```
# cat alert | grep '\[\*\*\]' | sort | uniq -c | sort -rn | cat >
alert.stats &
# cat alert.stats
     43 [**] [1:2003:2] MS-SQL Worm propagation attempt [**]
     24 [**] [1:615:3] SCAN SOCKS Proxy attempt [**]
```

A sample of the alert file:

```
[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[Classification: Misc Attack] [Priority: 2]
07/14-02:28:28.605996 66.199.149.229:1076 -> X.X.X.X:1434
UDP TTL:113 TOS:0x0 ID:20682 IpLen:20 DgmLen:404
Len: 376
[Xref      =>      http://vil.nai.com/vil/content/v_99992.htm][Xref      =>
http://www.securityfocus.com/bid/5311][Xref                               =>
http://www.securityfocus.com/bid/5310]

[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[Classification: Misc Attack] [Priority: 2]
07/14-02:29:23.288610 64.254.238.245:2876 -> X.X.X.X:1434
UDP TTL:107 TOS:0x0 ID:28892 IpLen:20 DgmLen:404
Len: 376
[Xref      =>      http://vil.nai.com/vil/content/v_99992.htm][Xref      =>
http://www.securityfocus.com/bid/5311][Xref                               =>
http://www.securityfocus.com/bid/5310]

[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[Classification: Misc Attack] [Priority: 2]
07/14-05:57:55.813146 62.174.168.131:3071 -> X.X.X.X:1434
UDP TTL:106 TOS:0x0 ID:16071 IpLen:20 DgmLen:404
Len: 376
[Xref      =>      http://vil.nai.com/vil/content/v_99992.htm][Xref      =>
http://www.securityfocus.com/bid/5311][Xref                               =>
http://www.securityfocus.com/bid/5310]

[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[Classification: Misc Attack] [Priority: 2]
07/14-06:00:22.066200 61.28.28.98:3676 -> X.X.X.X:1434
UDP TTL:112 TOS:0x0 ID:39966 IpLen:20 DgmLen:404
Len: 376
[Xref      =>      http://vil.nai.com/vil/content/v_99992.htm][Xref      =>
http://www.securityfocus.com/bid/5311][Xref                               =>
http://www.securityfocus.com/bid/5310]

[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[Classification: Misc Attack] [Priority: 2]
07/14-07:10:00.084010 212.103.162.176:1558 -> X.X.X.X:1434
UDP TTL:108 TOS:0x0 ID:4341 IpLen:20 DgmLen:404
Len: 376
[Xref      =>      http://vil.nai.com/vil/content/v_99992.htm][Xref      =>
http://www.securityfocus.com/bid/5311][Xref                               =>
http://www.securityfocus.com/bid/5310]

[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[Classification: Misc Attack] [Priority: 2]
07/14-08:13:55.508318 209.180.171.224:62694 -> X.X.X.X:1434
UDP TTL:108 TOS:0x0 ID:46224 IpLen:20 DgmLen:404
Len: 376
```

```
[Xref      =>      http://vil.nai.com/vil/content/v_99992.htm][Xref      =>
http://www.securityfocus.com/bid/5311][Xref                              =>
http://www.securityfocus.com/bid/5310]

[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[Classification: Misc Attack] [Priority: 2]
07/14-08:42:45.180883 67.232.141.219:1204 -> X.X.X.X:1434
UDP TTL:112 TOS:0x0 ID:50108 IpLen:20 DgmLen:404
Len: 376
[Xref      =>      http://vil.nai.com/vil/content/v_99992.htm][Xref      =>
http://www.securityfocus.com/bid/5311][Xref                              =>
http://www.securityfocus.com/bid/5310]
```

The triggering rule for the MS-SQL Worm propagation attempts was found to be:

```
sql.rules:alert udp $EXTERNAL_NET any -> $HOME_NET 1434 (msg:"MS-SQL Worm
propagation attempt"; content:"|04|"; depth:1; content:"|81 F1 03 01 04 9B
81  F1  01|";  content:"sock";  content:"send";  reference:bugtraq,5310;
classtype:misc-attack;reference:bugtraq,5311;reference:
url,vil.nai.com/vil/content/v_99992.htm; sid:2003; rev:2;)
```

Using *tcpdump*, I found the packet contents that triggered the alert:

```
# tcpdump -r snort_0714.1058688109 -nX 'udp and dst port 1434'
```

```
23:58:48.131367 195.29.54.131.3075 > X.X.X.X.1434: udp 376
0x0000   4500 0194 a210 0000 6b11 91fb c31d 3683      E.......k.....6.
0x0010   daba 45f2 0c03 059a 0180 0efb 0401 0101      ..E.............
0x0020   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0030   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0040   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0050   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0060   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0070   0101 0101 0101 0101 0101 0101 01dc c9b0      ................
0x0080   42eb 0e01 0101 0101 0101 70ae 4201 70ae      B.........p.B.p.
0x0090   4290 9090 9090 9090 9068 dcc9 b042 b801      B........h...B..
0x00a0   0101 0131 c9b1 1850 e2fd 3501 0101 0550      ...1...P..5....P
0x00b0   89e5 5168 2e64 6c6c 6865 6c33 3268 6b65      ..Qh.dllhel32hke
0x00c0   726e 5168 6f75 6e74 6869 636b 4368 4765      rnQhounthickChGe
0x00d0   7454 66b9 6c6c 5168 3332 2e64 6877 7332      tTf.llQh32.dhws2
0x00e0   5f66 b965 7451 6873 6f63 6b66 b974 6f51      _f.etQhsockf.toQ
0x00f0   6873 656e 64be 1810 ae42 8d45 d450 ff16      hsend....B.E.P..
0x0100   508d 45e0 508d 45f0 50ff 1650 be10 10ae      P.E.P.E.P..P....
0x0110   428b 1e8b 033d 558b ec51 7405 be1c 10ae      B....=U..Qt.....
0x0120   42ff 16ff d031 c951 5150 81f1 0301 049b      B....1.QQP......
0x0130   81f1 0101 0101 518d 45cc 508b 45c0 50ff      ......Q.E.P.E.P.
0x0140   166a 116a 026a 02ff d050 8d45 c450 8b45      .j.j.j...P.E.P.E
0x0150   c050 ff16 89c6 09db 81f3 3c61 d9ff 8b45      .P........<a...E
0x0160   b48d 0c40 8d14 88c1 e204 01c2 c1e2 0829      ...@...........)
0x0170   c28d 0490 01d8 8945 b46a 108d 45b0 5031      .......E.j..E.P1
0x0180   c951 6681 f178 0151 8d45 0350 8b45 ac50      .Qf..x.Q.E.P.E.P
0x0190   ffd6 ebca                                    ....
```

As quoted from http://vil.nai.com/vil/content/v_99992.htm[19],

"This virus exists only in memory of unpatched Microsoft SQL servers. Its purpose is simply to spread from one system to another and it does not carry a destructive payload. This worm causes increased traffic on UDP port 1434 and spreads between SQL servers. Heavy network traffic, associated with this threat, can effect network performance on all systems on the network. It uses a buffer overflow in "Server Resolution" service (read about CVE-CAN-2002-0649 vulnerability in MS02-39 and to gain control on a target server. SQL Servers running Service Pack 3 are not affected. The malformed packet is only 376 bytes long (which is the full worm!) and carries the following strings: "h.dllhel32hkernQhounthickChGetTf", "hws2", "Qhsockf" and "toQhsend"."

Based on the Snort signature, the alert was triggered by:
- UDP packet from external network
- First byte of UDP payload "0x04"
- Hex contents "0x81 0xf1 0x03 0x01 0x04 0x9b" in payload
- String "send" and "sock" in payload

## Probability the source address was spoofed

I tried to deduce where these worm packets originated from:

```
# mergecap -w snortdump snort_071*
# tcpdump -r snortdump -n 'udp and dst port 1434' | awk '{ print $2 }' |
sort | uniq -c | sort -rn | wc -l
      43
```

43 unique source addresses were accounted for, and this corresponded to the 43 MS-SQL worm alerts earlier. As these were UDP packets, it was trivial to generate them with spoofed addresses. However, I suspected these packets originated from compromised MS-SQL hosts.

I summarized the list of source addresses and its related TTL value:

| **#** | **Source address** | **Recorded TTL value From source (1)** | **Traceroute (no. of hops) from my IP (2)** | **Expected initial TTL value ( = (1) + (2) )** |
|---|---|---|---|---|
| 1 | 66.199.149.229 | 113 | 16 – reachable | 129 |
| 2 | 64.254.238.245 | 107 | 20 - unreachable | 127 |
| 3 | 62.174.168.131 | 106 | 24 - unreachable | 130 |
| 4 | 61.28.28.98 | 112 | 16 - reachable | 128 |
| 5 | 212.103.162.176 | 108 | 22 - unreachable | 130 |
| 6 | 209.180.171.224 | 108 | 20 - unreachable | 128 |
| 7 | 67.232.141.219 | 112 | 16 - unreachable | 128 |
| 8 | 68.98.153.220 | 111 | 22 - unreachable | 133 |
| 9 | 170.110.31.84 | 112 | 17 - reachable | 129 |
| 10 | 195.29.54.131 | 107 | 24 - reachable | 131 |
| 11 | 208.199.92.197 | 108 | 21 - reachable | 129 |
| 12 | 12.216.120.148 | 113 | 17 - unreachable | 130 |
| 13 | 80.164.77.19 | 114 | 22 - unreachable | 136 |
| 14 | 212.37.205.96 | 113 | 17 - reachable | 130 |
| 15 | 80.202.85.3 | 106 | 21 - unreachable | 127 |
| 16 | 24.136.203.109 | 108 | 21 - unreachable | 129 |
| 17 | 218.188.55.67 | 109 | 13 - unreachable | 122 |

| # | Source address | Recorded TTL value From source (1) | Traceroute (no. of hops) from my IP (2) | Expected initial TTL value ( = (1) + (2) ) |
|---|---|---|---|---|
| 18 | 195.61.75.183 | 106 | 21 - reachable | 127 |
| 19 | 217.97.53.217 | 106 | 22 - reachable | 128 |
| 20 | 203.155.234.8 | 110 | 17 - reachable | 127 |
| 21 | 195.174.71.148 | 114 | 21 - reachable | 135 |
| 22 | 129.19.3.8 | 111 | 16 - unreachable | 127 |
| 23 | 64.187.31.139 | 116 | 17 - reachable | 133 |
| 24 | 212.37.11.54 | 110 | 14 - unreachable | 125 |
| 25 | 213.227.150.150 | 109 | 22 - reachable | 131 |
| 26 | 203.107.147.186 | 111 | 20 - reachable | 131 |
| 27 | 64.187.61.110 | 116 | 17 - reachable | 133 |
| 28 | 80.200.47.145 | 108 | 17 - reachable | 125 |
| 29 | 200.43.11.97 | 109 | 19 - unreachable | 128 |
| 30 | 210.13.19.11 | 116 | 19 - reachable | 135 |
| 31 | 211.167.92.168 | 110 | 18 - reachable | 128 |
| 32 | 203.112.97.160 | 118 | 19 - reachable | 137 |
| 33 | 221.6.233.30 | 117 | 16 - reachable | 133 |
| 34 | 68.32.170.224 | 111 | 17 - unreachable | 128 |
| 35 | 64.38.236.45 | 113 | 17 - reachable | 130 |
| 36 | 202.108.220.187 | 111 | 17 - reachable | 128 |
| 37 | 212.19.31.2 | 99 | 24 - unreachable | 123 |
| 38 | 64.237.33.10 | 105 | 20 - reachable | 125 |
| 39 | 128.2.166.125 | 111 | 18 - reachable | 129 |
| 40 | 220.208.245.28 | 108 | 20 - reachable | 128 |
| 41 | 208.58.202.234 | 111 | 23 - reachable | 134 |
| 42 | 211.24.158.3 | 117 | 30 - reachable | 147 |
| 43 | 195.38.27.246 | 106 | 18 – reachable | 124 |

For those unreachable IP addresses, meaning the traceroute time-out before the max 30 hop-count is reached, I used *whois* to verify that the last responding hop-router belonged to the same organization as the IP. For example, consider #17:

```
# traceroute 218.188.55.67
traceroute to 218.188.55.67 (218.188.55.67), 30 hops max, 38 byte packets
 1  192.168.123.254 (192.168.123.254)  1.262 ms  1.598 ms  1.250 ms
 2  10.52.0.1 (10.52.0.1)  33.355 ms  17.612 ms  26.912 ms
 3  172.20.52.129 (172.20.52.129)  11.421 ms  15.634 ms  15.710 ms
 4  172.26.52.1 (172.26.52.1)  19.656 ms  11.541 ms  14.295 ms
 5  172.20.6.7 (172.20.6.7)  11.250 ms  19.695 ms  10.394 ms
 6  * * *
 7  203.118.3.203 (203.118.3.203)  18.029 ms  34.176 ms  24.734 ms
 8  134.159.125.65 (134.159.125.65)  23.032 ms  21.225 ms  14.708 ms
 9   i-2-0.ntp-core01.net.reach.com (202.84.180.141)  12.663 ms  13.125 ms
17.107 ms
10   i-5-7.tmhstcbr01.net.reach.com (202.84.249.213)  44.983 ms  47.576 ms
```

```
48.965 ms
11  i-10-0.tmhstcar01.net.reach.com (207.176.96.67)  46.666 ms  48.138 ms
45.591 ms
12  hutchcity2-RGE.hkix.net (202.40.161.114)  55.219 ms  48.101 ms  59.171
ms
13  210.0.248.222 (210.0.248.222)  45.692 ms  45.687 ms  50.235 ms
14  * * *
15  * * *
16  * * *
17  * * *
```

The last responding hop was IP 210.0.248.222. I did a *whois* on this address and my intended destination IP.

```
# whois -h whois.apnic.net 210.0.248.222
# whois -h whois.apnic.net 218.188.55.67
```

Both belonged to HTHKNET (Hutchinson Telecommunications, HK). Hence, give and take a few TTLs, I could, with confidence, estimate the IP was about 13 hops away. By adding this TTL value to the one recorded from the corresponding UDP packet, I was able to deduce the initial TTL value of the packet.

The expected TTL values revolved around 128, which suggests a Windows 9x/NT/2000 system generating the packets. Quite rightfully so, if my earlier take that these UDP packets originated from compromised hosts.

## Description of the attack

The worm in action has been given names like 'Sapphire', 'SQL-Hell' and 'MS-SQL Slammer'[20]. It first struck in Jan 2003 and has caused widespread damage not because of any malicious intent of the payload (like deletion of files) but rather the speed of propagation. A compromised MS-SQL host sends out UDP packets containing the worm payload of 376 bytes to port 1434 of randomly selected IP targets in an infinite loop. When unpatched MS-SQL servers (pre-SP3) receive these packets, the worm code receives control of the servers, and in turn propagates itself to other unpatched servers. The worm runs in the memory of the infected server.

## Attack mechanism

The Slammer worm based is a stack-based buffer overflow attack against the MS-SQL server. When a SQL Server receives a packet on UDP port 1434 with the first byte set to 0x04, the SQL Monitor thread takes the remaining data in the packet and attempts to open a registry key using this user supplied information. In the worm code, 0x04 was followed by a long series of 0x01, hence, the SQL Monitor generates a registry key:

HKLM\Software\Microsoft\Microsoft SQL Server\......\MSSQLServer\CurrentVersion where ..... represents unprintable 0x01 characters. This overflows the buffer, and the return address is overwritten, giving the worm control as well as privileges of the SQL Monitor. After which, the worm loads WS2_32.DLL (Winsock) and starts to propagate itself to UDP port 1434 of randomly selected IP addresses in an infinite loop.

A complete disassembly of the worm code can be found at http://www.nextgenss.com/advisories/mssql-udp.txt[21].

## Correlations

The Slammer worm was first detected in late Jan 2003 and recorded as the fastest spreading worm in history. The capture of the worm in the wild about 6-months later showed that there are still some compromised MS-SQL hosts out there attempting to propagate the worm. Nothing else, I presumed, since none of them featured in Block List found at http://www.dshield.org/block_list_info.php.

## Evidence of active targeting

The worm propagates itself by sending the UDP packets to randomly selected IP addresses. I also observed the source port used was ephemeral (>1024). A common, firewall-friendly source port such as 53 would indicate an intentional effort to punch the UDP packets through. But not so in this case. Hence, I would think that no active targeting was involved.

## Severity

**Criticality = 1**
Criticality is low because there was no host involved. The analysis was carried out on raw traffic captured off a Cable Internet connection.

**Lethality = 3**
Lethality was high if an unpatched MS-SQL server was connected to the network.

**System countermeasures = 1**
No concerns since my router/firewall did not listen on any services or ports.

**Network countermeasures = 1**
My router/firewall was configured to drop any incoming UDP packets.

**Severity = (1+3) – (1+1) = 2**

## Defensive recommendations

My router diligently dropped these attack packets from reaching into my network. I had deliberately set up the Linux box for traffic collection, hence I would not be too concerned with any other measures to defend against this attack.

## Multiple choice question

What is one of the possible ways to deduce whether a packet originated from a spoofed IP address?

a. Ping the source IP
b. Traceroute to the source IP
c. Estimate the initial TTL value of the packet and check whether it matches the signature of any OS
d. Look up *Dshield*

The answer is (c).

# Part 3: Analyze this!

## 1. Executive summary:

This document summarizes the observations and analysis of 5 days' worth of logs files collected from your organization. Alerts, scans and OOS logs from 9th to 13th Jul 2003 were analyzed.

Several Snort alerts were detected during the analysis period. Link graphs were constructed to give a better picture of the relationships between the various machines and the type of activities. There were a few instances of suspicious activity involving computers from your network. Such activities may indicate signs of compromise, like the use of compromised machines to attack other networks. And even leakage of information about your network to potential attackers.

Alerts found in your network were compared against the Top 20 critical Internet Security vulnerabilities according to SANS/FBI, giving an indication of the threat to your organization. A brief description of the other alerts were also provided. Five external addresses with their registration information were highlighted.

Throughout the document, defensive recommendations were provided for your consideration.

## 2. Log files used in analysis:

The logs from 9th to 13th Jul 2003 were downloaded from your IDS for analysis. The 15 files were:

```
$ ls -l
total 965504
-rwxrwxr-x    1 johnwong johnwong 27247358 Aug  2 09:51 alert.030709
-rwxrwxr-x    1 johnwong johnwong 35036701 Aug  2 09:53 alert.030710
-rwxrwxr-x    1 johnwong johnwong 33882668 Aug  2 09:55 alert.030711
-rwxrwxr-x    1 johnwong johnwong 42468295 Aug  2 09:55 alert.030712
-rwxrwxr-x    1 johnwong johnwong 29950716 Aug  2 09:57 alert.030713
-rwxrwxr-x    1 johnwong johnwong  1029123 Aug  2 09:51 OOS_Report_2003_07_09_2126
-rwxrwxr-x    1 johnwong johnwong  1402883 Aug  2 09:51 OOS_Report_2003_07_10_4402
-rwxrwxr-x    1 johnwong johnwong  1136643 Aug  2 09:51 OOS_Report_2003_07_11_27931
-rwxrwxr-x    1 johnwong johnwong  7255043 Aug  2 09:55 OOS_Report_2003_07_12_20109
-rwxrwxr-x    1 johnwong johnwong  6594563 Aug  2 09:55 OOS_Report_2003_07_13_9896
-rwxrwxr-x    1 johnwong johnwong 197186311 Aug  2 10:00 scans.030709
-rwxrwxr-x    1 johnwong johnwong 236729582 Aug  2 10:01 scans.030710
```

-rwxrwxr-x 1 johnwong johnwong 122446016 Aug 2 10:00 scans.030711
-rwxrwxr-x 1 johnwong johnwong 120677834 Aug 2 10:01 scans.030712
-rwxrwxr-x 1 johnwong johnwong 125366456 Aug 2 10:01 scans.030713

Each type of files were concatenated into a single file: alert-all.txt, OOS_Report-all.txt and scans-all.txt. As also observed by Fred Thiele's GCIA submission[22], there were some discrepancies observed in the alert file:

- Alerts were interjected in the middle of another. For instance,

  **07/09-00:59:21.716736 [**] CS WEBSERVER - external web traffic [**] 66.196.72.7**
  **3**07/09-01:36:06.603290 [**] spp_portscan: portscan status from MY.NET.1.4: 9 co
  nnections across 9 hosts: TCP(0), UDP(9) [**]
  07/09-01:36:06.606318 [**] spp_portscan: portscan status from MY.NET.114.45: 75
  connections across 75 hosts: TCP(75), UDP(0) [**]
  07/09-01:36:06.700015 [**] spp_portscan: portscan status from MY.NET.1.3: 3 con
  nections across 3 hosts: TCP(0), UDP(3) [**]
  07/09-01:36:06.700560 [**] spp_portscan: portscan status from MY.NET.1.4: 14 co
  nnections across 14 hosts: TCP(0), UDP(14) [**]
  **:28765 -> MY.NET.100.165:80**
  The alert in **BOLD RED** was split in the middle by the portscan entries.

- The alert file also contained a large amount of spp_portscan entries. These were filtered off into another file (alert-portscan-all.txt) for ease of analysis. The main file was alert-all.filtered.txt.

A manual clean-up of the files was carried out before the analysis began, which took me about 2 days.

## 3. Pre-processing of alert files and tools used:

A combination of Unix command line tools like sed, cut, awk and grep were used to massage the raw alerts, scans and OOS files into a format compatible for entry into MySQL databases. Refer to Annex A for the details.

## 4. Summary of alerts:

A total of 589,595 alerts were generated during this 5-day period, which excluded the port scan alerts. There were 61 unique alerts identified. The alerts were:

| # | Alert | #src | #dst | Occurrence |
|---|---|---|---|---|
| 1 | High port 65535 tcp - possible Red Worm - traffic | 108 | 132 | 133659 |
| 2 | CS WEBSERVER - external web traffic | 20657 | 6 | 128355 |
| 3 | spp_http_decode- IIS Unicode attack detected | 550 | 1193 | 98201 |
| 4 | SMB Name Wildcard | 894 | 1296 | 70614 |
| 5 | MY.NET.30.4 activity | 521 | 4 | 44549 |
| 6 | SYN-FIN scan! | 7 | 23369 | 36271 |
| 7 | EXPLOIT x86 NOOP | 62 | 88 | 32702 |
| 8 | MY.NET.30.3 activity | 78 | 1 | 9146 |
| 9 | spp_http_decode- CGI Null Byte attack detected | 88 | 133 | 8995 |
| 10 | Queso fingerprint | 341 | 95 | 8861 |
| 11 | Null scan! | 47 | 47 | 2402 |

| | | | | |
|---|---|---|---|---|
| 12 | connect to 515 from inside | 6 | 4 | 1973 |
| 13 | IDS552/web-iis_IIS ISAPI Overflow ida nosize | 881 | 466 | 1566 |
| 14 | TCP SRC and DST outside network | 112 | 434 | 1530 |
| 15 | connect to 515 from outside | 1 | 1 | 1384 |
| 16 | High port 65535 udp - possible Red Worm - traffic | 63 | 56 | 1311 |
| 17 | FTP passwd attempt | 46 | 80 | 1283 |
| 18 | [UMBC NIDS IRC Alert] IRC user /kill detected, po | 53 | 58 | 1161 |
| 19 | External RPC call | 6 | 590 | 989 |
| 20 | IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL no | 2 | 515 | 827 |
| 21 | NMAP TCP ping! | 134 | 68 | 726 |
| 22 | Possible trojan server activity | 52 | 61 | 527 |
| 23 | NIMDA - Attempt to execute cmd from campus host | 6 | 393 | 403 |
| 24 | Incomplete Packet Fragments Discarded | 57 | 40 | 332 |
| 25 | Tiny Fragments - Possible Hostile Activity | 3 | 0 | 276 |
| 26 | SNMP public access | 1 | 1 | 156 |
| 27 | SUNRPC highport access! | 17 | 16 | 150 |
| 28 | SMB C access | 68 | 9 | 139 |
| 29 | TFTP - Internal TCP connection to external tftp s | 6 | 41 | 118 |
| 30 | CS WEBSERVER - external ftp traffic | 19 | 1 | 108 |
| 31 | TCP SMTP Source Port traffic | 2 | 53 | 96 |
| 32 | Notify Brian B. 3.54 tcp | 45 | 1 | 91 |
| 33 | Notify Brian B. 3.56 tcp | 38 | 1 | 79 |
| 34 | TFTP - Internal UDP connection to external tftp s | 7 | 10 | 64 |
| 35 | EXPLOIT x86 stealth noop | 6 | 6 | 64 |
| 36 | MYPARTY - Possible My Party infection | 1 | 1 | 50 |
| 37 | EXPLOIT x86 setuid 0 | 29 | 28 | 49 |
| 38 | RFB - Possible WinVNC - 010708-1 | 17 | 23 | 49 |
| 39 | NETBIOS NT NULL session | 6 | 8 | 47 |
| 40 | DDOS shaft client to handler | 8 | 4 | 39 |
| 41 | EXPLOIT x86 setgid 0 | 29 | 24 | 34 |
| 42 | EXPLOIT NTPDX buffer overflow | 6 | 7 | 25 |
| 43 | TFTP - External TCP connection to internal tftp s | 6 | 8 | 22 |
| 44 | Probable NMAP fingerprint attempt | 5 | 8 | 21 |
| 45 | Attempted Sun RPC high port access | 6 | 7 | 17 |
| 46 | IRC evil - running XDCC | 3 | 3 | 16 |
| 47 | FTP DoS ftpd globbing | 1 | 1 | 16 |
| 48 | [UMBC NIDS IRC Alert] Possible sdbot floodnet det | 12 | 1 | 16 |
| 49 | External FTP to HelpDesk MY.NET.70.49 | 7 | 1 | 15 |
| 50 | External FTP to HelpDesk MY.NET.70.50 | 6 | 1 | 14 |
| 51 | ICMP SRC and DST outside network | 6 | 0 | 10 |
| 52 | DDOS mstream handler to client | 1 | 3 | 8 |
| 53 | External FTP to HelpDesk MY.NET.53.29 | 5 | 1 | 8 |
| 54 | Back Orifice | 2 | 3 | 7 |
| 55 | TFTP - External UDP connection to internal tftp s | 3 | 5 | 6 |
| 56 | Traffic from port 53 to port 123 | 2 | 2 | 5 |
| 57 | DDOS mstream client to handler | 2 | 2 | 3 |
| 58 | [UMBC NIDS IRC Alert] User joining Warez channel | 2 | 2 | 3 |
| 59 | [UMBC NIDS IRC Alert] Possible Incoming XDCC Send | 3 | 3 | 3 |
| 60 | [UMBC NIDS IRC Alert] K | 1 | 1 | 2 |
| 61 | NIMDA - Attempt to execute root from campus host | 1 | 2 | 2 |

**Table 1** Consolidated table of alerts collected from 9th to 13th Jul 2003

Some of the alert signatures were most likely created locally, such as #5, #8, #32, #33, #49 and #50. For example, #5 was triggered whenever there was activity involving the host MY.NET.30.4. Inbound web traffic would trigger #2. The Snort signatures were probably crafted as such:

alert any any -> MY.NET.0.0/16 80 (msg: "CS WEBSERVER – external web traffic"; )
alert any any -> MY.NET.30.3/32 any (msg: "MY.NET.30.3 activity"; )
alert any any -> MY.NET.30.4/32 any (msg: "MY.NET.30.4 activity"; )

The IDS was probably sited at the edge of your campus network in between the perimeter router and firewall.

Other statistics worth noting were:

| | |
|---|---|
| **Total number of alerts detected** | **589,595** |
| **Total number of unique alerts** | **61** |
| **Total number of unique source addresses detected** | **24,428** |
| **- from MY.NET** | 466 |
| **- from external** | 23,962 |
| **Total number of unique destination addresses detected** | **25,687** |
| **- from MY.NET** | 23,635 |
| **- from external** | 2,052 |
| **Total number of unique attacks launched by MY.NET hosts** | **18** |
| **Total number of unique attacks launched against MY.NET hosts** | **49** |

A total of 589,595 alerts were triggered by 24,428 hosts (internal and external), which indicates many of the attacks originated from the same host. Most of the attacks originated from external addresses, which made up 98% of the total number of attack-related source addresses. A total of 23,635 hosts from your network were targeted, which made up 92% of the total number attacked destination addresses.

The previous figures indicated that a number of hosts in your network were used to attack other networks, or might be compromised and used as launch-pads for attacks against other networks. 18 unique types of attack were found. However, a larger number of hosts in your network were attacked. 49 unique attacks were launched against the hosts in your network.

## 5. Relationships between the various addresses

The following tables were created to look for any meaningful relationships between the various addresses:

**Attacks involving MY.NET hosts**

| **Top 10 source from MY.NET** | | | |
|---|---|---|---|
| **Address** | **Type of alerts triggered** | **Occurrence per type of alert** | **% of total alerts generated by MY.NET hosts** |
| MY.NET.82.36 | Port 65536 tcp-Red Worm | 78692 | 48.28% |
| | CGI Null Byte | 61 | |

| | | | |
|---|---|---|---|
| | IIS Unicode | 2 | |
| MY.NET.198.172 | IIS Unicode | 9735 | 5.97% |
| MY.NET.153.185 | IIS Unicode | 7297 | 4.47% |
| MY.NET.97.79 | IIS Unicode | 5376 | 3.49% |
| | Conn to 515 from inside | 313 | |
| MY.NET.97.168 | IIS Unicode | 3470 | 2.13% |
| MY.NET.97.63 | IIS Unicode | 3120 | 1.91% |
| MY.NET.153.211 | IIS Unicode | 2398 | 1.47% |
| MY.NET.84.216 | IIS Unicode | 2096 | 1.28% |
| MY.NET.97.44 | IIS Unicode | 1678 | 1.03% |
| MY.NET.97.64 | IIS Unicode | 1667 | 1.03% |
| | Port 65536 tcp-Red Worm | 6 | |

**Table 2**

**Observation:**
The host MY.NET.82.36 triggered the most number of alerts, 99.9% of which came from Red Worm attacks. It was also noted that **all** the top 10 MY.NET hosts triggered the IIS Unicode alert.

**Top destination from MY.NET**

| Address | # of unique type of alerts triggered | Occurrence per type of alert | % of total alerts generated by MY.NET hosts |
|---|---|---|---|
| MY.NET.100.165 | CS WEBSERVER – ext web | 128346 | 30.34% |
| | Others (11 types) | 468 | |
| MY.NET.82.36 | Port 65536 tcp-Red Worm | 53783 | 12.67% |
| | Others (4 types) | 7 | |
| MY.NET.30.4 | 6 types | 44612 | 10.51% |
| MY.NET.30.3 | 4 types | 9215 | 2.17% |
| MY.NET.24.8 | 5 types | 5606 | 1.32% |
| MY.NET.137.7 | 6 types | 5518 | 1.30% |
| MY.NET.137.46 | 5 types | 3451 | 0.81% |
| MY.NET.86.19 | 5 types | 2212 | 0.52% |
| MY.NET.5.67 | 4 types | 1973 | 0.46% |
| MY.NET.189.62 | 5 types | 1963 | 0.46% |

**Table 3**

**Observation:**
MY.NET.100.165 was the most attacked host in your network. MY.NET.82.36, which occupied the top spot in Table 2, also appeared here, in second place. Strange enough, the Red Worm attack which was previously used by this host was now used against it here.

**Type of attacks launched by MY.NET hosts**

| # | Alert | # of MY.NET hosts involved | Occurrence |
|---|---|---|---|
| 1 | High port 65535 tcp - possible Red Worm - traffic | 38 | 79392 |
| 2 | spp_http_decode- IIS Unicode attack detected | 368 | 70623 |
| 3 | spp_http_decode- CGI Null Byte attack detected | 81 | 8846 |

| | | | |
|---|---|---|---|
| 4 | connect to 515 from inside | 6 | 1973 |
| 5 | IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL no | 2 | 827 |
| 6 | High port 65535 udp - possible Red Worm - traffic | 11 | 496 |
| 7 | NIMDA - Attempt to execute cmd from campus host | 6 | 403 |
| 8 | Possible trojan server activity | 21 | 272 |
| 9 | Incomplete Packet Fragments Discarded | 1 | 123 |
| 10 | MYPARTY - Possible My Party infection | 1 | 50 |
| 11 | TFTP - Internal TCP connection to external tftp s | 2 | 29 |
| 12 | RFB - Possible WinVNC - 010708-1 | 8 | 27 |
| 13 | TFTP - Internal UDP connection to external tftp s | 4 | 25 |
| 14 | [UMBC NIDS IRC Alert] Possible sdbot floodnet det | 12 | 16 |
| 15 | IRC evil - running XDCC | 3 | 16 |
| 16 | DDOS mstream handler to client | 1 | 8 |
| 17 | TFTP - External TCP connection to internal tftp s | 4 | 6 |
| 18 | NIMDA - Attempt to execute root from campus host | 1 | 2 |

**Table 4**

**Observation:**
A majority of MY.NET hosts triggered IIS Unicode alerts (#2). Some of the alerts could be due to false positives such as the IIS Unicode and CGI Null Byte attacks. #8, #10 and #16 are related to Trojan activities. #2, #3, #5, #7 and #18 are closely related to IIS vulnerabilities.

**Type of attacks launched against MY.NET hosts**

| | **Alert** | **# of MY.NET hosts targeted** | **Occurrence** |
|---|---|---|---|
| 1 | CS WEBSERVER - external web traffic | 5 | 128350 |
| 2 | SMB Name Wildcard | 1294 | 70612 |
| 3 | High port 65535 tcp - possible Red Worm - traffic | 44 | 54270 |
| 4 | MY.NET.30.4 activity | 4 | 44548 |
| 5 | SYN-FIN scan! | 23368 | 36268 |
| 6 | EXPLOIT x86 NOOP | 86 | 32700 |
| 7 | spp_http_decode- IIS Unicode attack detected | 450 | 27579 |
| 8 | MY.NET.30.3 activity | 1 | 9146 |
| 9 | Queso fingerprint | 94 | 8860 |
| 10 | Null scan! | 47 | 2402 |
| 11 | IDS552/web-iis_IIS ISAPI Overflow ida nosize | 466 | 1566 |
| 12 | connect to 515 from outside | 1 | 1384 |
| 13 | FTP passwd attempt | 80 | 1283 |
| 14 | [UMBC NIDS IRC Alert] IRC user /kill detected, po | 58 | 1161 |
| 15 | External RPC call | 590 | 989 |
| 16 | High port 65535 udp - possible Red Worm - traffic | 28 | 815 |
| 17 | NMAP TCP ping! | 68 | 726 |
| 18 | Possible trojan server activity | 20 | 255 |
| 19 | Incomplete Packet Fragments Discarded | 38 | 209 |
| 20 | SNMP public access | 1 | 156 |
| 21 | SUNRPC highport access! | 16 | 150 |
| 22 | spp_http_decode- CGI Null Byte attack detected | 7 | 149 |
| 23 | SMB C access | 9 | 139 |
| 24 | CS WEBSERVER - external ftp traffic | 1 | 108 |

| | | | |
|---|---|---|---|
| 25 | TCP SMTP Source Port traffic | 53 | 96 |
| 26 | Notify Brian B. 3.54 tcp | 1 | 91 |
| 27 | TFTP - Internal TCP connection to external tftp s | 39 | 89 |
| 28 | Notify Brian B. 3.56 tcp | 1 | 79 |
| 29 | EXPLOIT x86 stealth noop | 6 | 64 |
| 30 | EXPLOIT x86 setuid 0 | 28 | 49 |
| 31 | NETBIOS NT NULL session | 8 | 47 |
| 32 | DDOS shaft client to handler | 4 | 39 |
| 33 | TFTP - Internal UDP connection to external tftp s | 5 | 39 |
| 34 | EXPLOIT x86 setgid 0 | 24 | 34 |
| 35 | EXPLOIT NTPDX buffer overflow | 7 | 25 |
| 36 | RFB - Possible WinVNC - 010708-1 | 11 | 22 |
| 37 | Probable NMAP fingerprint attempt | 8 | 21 |
| 38 | Attempted Sun RPC high port access | 7 | 17 |
| 39 | TFTP - External TCP connection to internal tftp s | 6 | 16 |
| 40 | FTP DoS ftpd globbing | 1 | 16 |
| 41 | External FTP to HelpDesk MY.NET.70.49 | 1 | 15 |
| 42 | External FTP to HelpDesk MY.NET.70.50 | 1 | 14 |
| 43 | External FTP to HelpDesk MY.NET.53.29 | 1 | 8 |
| 44 | Back Orifice | 3 | 7 |
| 45 | TFTP - External UDP connection to internal tftp s | 5 | 6 |
| 46 | Traffic from port 53 to port 123 | 2 | 5 |
| 47 | [UMBC NIDS IRC Alert] User joining Warez channel | 2 | 3 |
| 48 | DDOS mstream client to handler | 2 | 3 |
| 49 | [UMBC NIDS IRC Alert] Possible Incoming XDCC Send | 3 | 3 |

**Table 5**

**Observation:**
A large number of MY.NET hosts were probed with SYN-FIN scans (#5). There were a couple of locally created alert signatures such as #1, #4 and #8. These were most likely used to track host activity.

## Attacks involving External hosts

| **Top 10 source from external** | | | |
|---|---|---|---|
| **Address** | **Type of alerts triggered** | **Occurrence per type of alert** | **% of total alerts generated by external hosts** |
| 24.84.205.243 | Port 65536 tcp-Red Worm | 53783 | 12.61% |
| 80.204.44.179 | IIS Unicode | 26387 | 6.37% |
| | 5 other types | 779 | |
| 142.26.120.7 | SYN-FIN scan | 20538 | 4.82% |
| 68.54.93.211 | MY.NET.30.4 activity | 18881 | 4.43% |
| 195.5.55.32 | SYN-FIN scan | 15722 | 3.69% |
| 172.176.163.24 | EXPLOIT x86 NOOP | 10828 | 2.54% |
| 169.254.45.176 | SMB Name Wildcard | 8084 | 1.90% |
| 217.88.160.45 | EXPLOIT x86 NOOP | 6620 | 1.55% |
| 172.180.87.233 | 6 types of alerts | 6571 | 1.54% |
| 131.118.254.13 | 4 types of alerts | 5352 | 1.25% |

**Table 6**

**Observation:**

The host at 24.84.205.243 featured prominently and closely related to the Red Worm attacks. Hosts 142.26.120.7 and 195.5.55.32 carried out SYN-FIN probes against machines in MY.NET, an observation noted earlier too (refer to Table 5). Hosts 172.168.163.24 and 217.88.160.45 contributed to most of the EXPLOIT x86 NOOP attacks against MY.NET machines.

| **Top 10 destination from external** | | | |
|---|---|---|---|
| **Address** | **Type of alerts triggered** | **Occurrence per type of alert** | **% of total alerts generated by external hosts** |
| 24.84.205.243 | Port 65536 tcp-Red Worm | 78690 | 47.79% |
| | SMB Name Wildcard | 1 | |
| 211.147.7.47 | IIS Unicode | 11598 | 7.04% |
| 210.192.111.73 | IIS Unicode | 9627 | 5.85% |
| | CS WEBSERVER – ext web | 1 | |
| 65.127.129.10 | IIS Unicode | 3750 | 2.28% |
| 207.200.86.97 | IIS Unicode | 2679 | 1.63% |
| 216.241.219.22 | CGI Null byte | 2247 | 1.36% |
| 218.153.6.229 | IIS Unicode | 2189 | 1.33% |
| 202.103.69.100 | IIS Unicode | 2044 | 1.24% |
| 218.153.6.244 | IIS Unicode | 1716 | 1.04% |
| 211.43.210.143 | IIS Unicode | 1536 | 0.93% |

**Table 7**

**Observation:**
Notice the similarity to Table 2. Most of the top external hosts targeted by IIS Unicode attacks.

| **Top source/alert pair** | | |
|---|---|---|
| Address | Alert | Occurrence |
| MY.NET.82.36 | High port 65535 tcp - possible Red Worm - traffic | 78692 |
| 24.84.205.243 | High port 65535 tcp - possible Red Worm - traffic | 53783 |
| 80.204.44.179 | spp_http_decode- IIS Unicode attack detected | 26387 |
| 142.26.120.7 | SYN-FIN scan! | 20538 |
| 68.54.93.211 | MY.NET.30.4 activity | 18881 |
| 195.5.55.32 | SYN-FIN scan! | 15722 |
| 172.176.163.24 | EXPLOIT x86 NOOP | 10828 |
| MY.NET.198.172 | spp_http_decode- IIS Unicode attack detected | 9735 |
| 169.254.45.176 | SMB Name Wildcard | 8084 |
| MY.NET.153.185 | spp_http_decode- IIS Unicode attack detected | 7297 |

**Table 8**

| **Top destination/alert pair** | | |
|---|---|---|
| Address | Alert | Occurrence |
| MY.NET.100.165 | CS WEBSERVER - external web traffic | 128346 |
| 24.84.205.243 | High port 65535 tcp - possible Red Worm - traffic | 78690 |
| MY.NET.82.36 | High port 65535 tcp - possible Red Worm - traffic | 53782 |
| MY.NET.30.4 | MY.NET.30.4 activity | 44545 |
| 211.147.7.47 | spp_http_decode- IIS Unicode attack detected | 11598 |
| 210.192.111.73 | spp_http_decode- IIS Unicode attack detected | 9627 |
| MY.NET.30.3 | MY.NET.30.3 activity | 9146 |

| | | |
|---|---|---|
| MY.NET.24.8 | EXPLOIT x86 NOOP | 5520 |
| MY.NET.137.7 | SMB Name Wildcard | 5437 |
| 65.127.129.10 | spp_http_decode- IIS Unicode attack detected | 3750 |

**Table 9**

| **Top source-destination pair** | | |
|---|---|---|
| Source | Destination | Occurrence |
| MY.NET.82.36 | 24.84.205.243 | 78688 |
| 24.84.205.243 | MY.NET.82.36 | 53780 |
| 68.54.93.211 | MY.NET.30.4 | 18881 |
| MY.NET.198.172 | 210.192.111.73 | 9627 |
| 131.118.254.13 | MY.NET.24.8 | 5352 |
| MY.NET.97.79 | 211.147.7.47 | 5349 |
| 193.41.146.24 | MY.NET.100.165 | 4576 |
| 65.214.36.116 | MY.NET.100.165 | 3760 |
| MY.NET.97.168 | 65.127.129.10 | 3470 |
| 68.55.52.234 | MY.NET.30.3 | 3182 |

**Table 10**

**Observation:**
Notice the two-way relationship between MY.NET.82.36 and 24.84.205.243, both directions involved the Red Worm attacks. The external host 210.192.111.73 was one of the top listener and all its traffic came from MY.NET.198.172. The alerts that came from locally created signatures also featured prominently, namely the CS WEBSERVER and MY.NET.30.4 activity. Note the high occurrence of web traffic from 193.41.146.24 and 65.214.36.116.

Note also the high frequency of SYN-FIN and EXPLOIT x86 NOOP in Table 8, but none of the source addresses featured in Table 10. This was because they were probes sent to a range of destination addresses, not active targeting of a single host.

## Reviewing the scans logs

There were a total of 12,281,498 scans detected during the five-day period from 9th to 13th Jul. From the "scans" files, the "MY.NET" prefixes were replaced with "130.85", which coincidently owned by Maryland University. The Top scans activities by to protocol type were:

| **Top 5 Scans type** | | | |
|---|---|---|---|
| **Type** | **# of unique source address** | **Count** | **Top destination port** |
| UDP | 576 | 6218981 | 53 (51%) |
| SYN | 943 | 5845119 | 80 (71%) |
| FIN | 31 | 177659 | 1214 (100%) |
| SYNFIN | 9 | 36281 | 21 (100%) |
| NULL | 49 | 889 | 110 (46%) |

**Table 11**

The other statistics generated from Table 11 were:

**Top Source Address to Destination UDP Port 53**

| Address | Occurrence | # of destination addresses |
|---|---|---|
| 130.85.1.3 | 2,698,955 | 72,208 |
| 130.85.1.4 | 468,395 | 30,475 |

These two hosts were sending a lot of DNS queries to a large of external hosts. Could these 2 be compromised servers sending out probes? A search in the alert logs of any alerts triggered with either of these 2 addresses as source turned out empty. This could be legitimate activity but I would still recommend checking out the contents of the UDP payloads sent out.

**Top Source Address of SYNFIN scans**

| Address | Occurrence |
|---|---|
| 142.26.120.7 | 20,538 |
| 195.5.55.32 | 15,723 |

The SYNFIN type statistics here complemented Table 8, identifying the top 2 culprits of SYN-FIN scans against your network to be 142.26.120.7 and 195.5.55.32. The SYN-FIN scans were targeted at port 21 (ftp).

**Possible scan activity from 130.85.114.45**

The host 130.85.114.45 was sending out SYN probes to a series of external addresses:

```
Jul  9 00:00:06 130.85.114.45:1042 -> 134.203.218.29:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1043 -> 134.203.218.30:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1044 -> 134.203.218.31:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1046 -> 134.203.218.32:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1047 -> 134.203.218.33:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1048 -> 134.203.218.34:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1049 -> 134.203.218.35:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1050 -> 134.203.218.36:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1051 -> 134.203.218.37:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1052 -> 134.203.218.38:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1053 -> 134.203.218.39:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1054 -> 134.203.218.40:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1056 -> 134.203.218.41:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1059 -> 134.203.218.42:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1060 -> 134.203.218.43:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1061 -> 134.203.218.44:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1062 -> 134.203.218.46:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1063 -> 134.203.218.45:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1065 -> 134.203.218.48:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1066 -> 134.203.218.49:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1067 -> 134.203.218.50:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1068 -> 134.203.218.51:80 SYN ******S*
Jul  9 00:00:06 130.85.114.45:1069 -> 134.203.218.52:80 SYN ******S*
```

Notice the traits of a portscan tool, such as incrementing source port with every next destination IP (sequential), and the frequency of the SYN packets being generated.

1,403,124 addresses were SYN-scanned by this host.

**Reviewing the OOS logs**

A total of 63,922 OOS packets (49 types) were observed throughout the 5-day period. The most common type of OOS packets were:

| OOS Type | No. of OOS packets | Percentage of total OOS packets | Remarks |
|---|---|---|---|
| ******SF | 48,039 | 75.15% | The top source of these OOS packets were 142.26.120.7 and 195.5.55.32, which complemented Table 8, the source IP of SYN-FIN scans. |
| 12****S* | 14,751 | 23.08% | Reference to the Queso fingerprint attack discussed later. |
| ******** | 638 | 1.00% | A signature of a NULL scan – see later section on detects. |

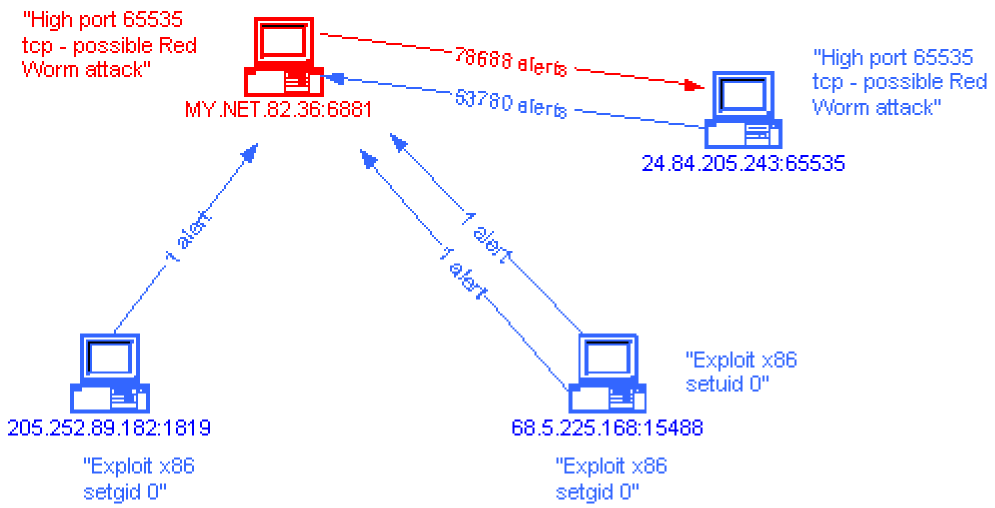
## 6. Link Graphs

Case 1 – Attacks involving the Red Worm
First, I examined the relationships between the hosts involved in the Red Worm attacks.

Not indicated in the diagram below, the host MY.NET.82.36 initiated some access to 217.215.27.135 port 80 from 1334hrs to 1510hrs on 07/10/03, which triggered 61 "CGI Null Byte detected" alerts. On 07/11/03, hosts 68.5.225.168 and 205.252.89.182 initiated connections to the MY.NET host port 6881, triggering the "Exploit x86 setuid 0" and "Exploit x86 setgid 0" alerts:

```
07/11-09:09:09.347592 | EXPLOIT x86 setuid 0 | 68.5.225.168 | 15448 | MY.NET.82.36 | 6881 |
07/11-09:23:47.215387 | EXPLOIT x86 setgid 0 | 68.5.225.168 | 15448 | MY.NET.82.36 | 6881 |
07/11-18:25:28.121629 | EXPLOIT x86 setgid 0 | 205.252.89.182 | 1819 | MY.NET.82.36 | 6881 |
```

The observation of the destination port targeted (6881) was significant, because on the following day, the traffic between MY.NET.82.36:**6881** and 24.84.205.243:**65535** triggered a barrage of Red Worm alerts on the same day. The host 24.84.205.243 did not communicate with any other MY.NET host. Was it purely coincidental or somehow the attack on the earlier day was related? A search of http://www.sans.org/y2k/ports.htm[23] did not return any associated service with this port number. Could MY.NET.82.36 be used to attack another host?




**Link Graph 1**

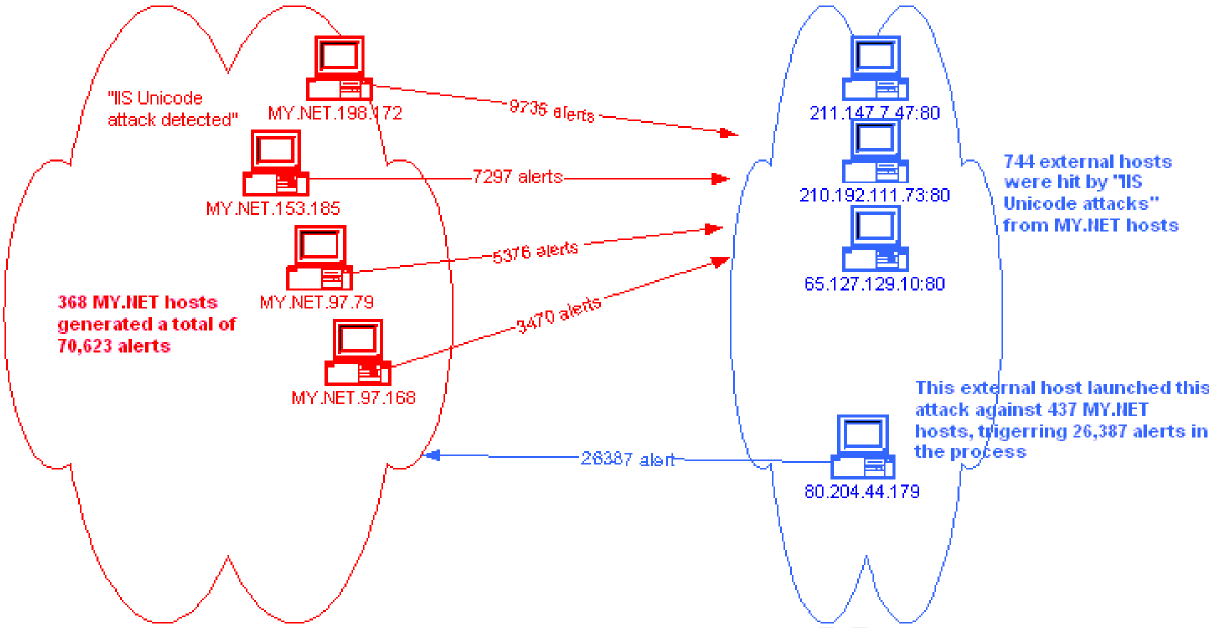
I would recommend checking on the machine MY.NET.82.36 for any signs of compromise. The raw logs could be examined to detect any malicious code in the payload.

<u>Case 2 – Attacks involving the IIS Unicode</u>

The "IIS Unicode attack" rated as number 3 overall for the number of alerts generated. As evident from Tables 2 and 7, there was a correlation between the "IIS Unicode attack" alerts triggered by the MY.NET hosts as source and the external hosts as destination. I also noted the external host 80.204.44.179 triggered the alert against 26,738 MY.NET hosts.

Let's look at the targeted external hosts first. 7 out of the top 10 targeted hosts belonged to sites located in Asia. Netscape.com also appeared in the list:

| External host | Location? | Alert Count | # of MY.NET hosts involved |
|---|---|---|---|
| 211.147.7.47 | China | 11,598 | 14 |
| 210.192.111.73 | China | 9,627 | 1 |
| 65.127.129.10 | U.S. - performancestore.com | 3,750 | 2 |
| 207.200.86.97 | U.S. Netscape.com | 2,679 | 6 |
| 218.153.6.229 | Korea | 2,189 | 1 |
| 202.103.69.100 | China | 2,044 | 10 |
| 218.153.6.244 | Korea | 1,716 | 1 |
| 211.43.210.143 | Korea | 1,536 | 1 |
| 66.36.238.12 | U.S. Mixedrace.com | 1,511 | 4 |
| 210.115.150.102 | Korea | 1,315 | 2 |




**Link Graph 2**

This attack is susceptible to generating false positives. Most Asian web sites use Unicode to display the language characters, such as Japanese, Korean and Chinese. These alerts might have been triggered as such. For the non-Asian sites, I would advise inspecting the payload for any malicious content. If the result of the inspection pointed to false positive, then I recommend turning off the detection of Unicode in the Snort IDS. The steps can be found at http://www.snort.org/docs/FAQ.txt[24] under item 4.17.

My attention now turned to the external host 80.204.44.179. This particular host launched the attack against 437 machines in your network, triggering 26,387 alerts in the process. This attack was launched against port 80 of the destination host. Hence, the locally created alerts such as:

- "MY.NET.30.4 activity",
- "MY.NET.30.3 activity",
- "CS WEBSERVER – external web traffic",
- "Notify Brian B. 3.54 tcp" and
- "Notify Brian B. 3.56 tcp"

were also triggered. The attack started on 07/10 at 0148hrs and lasted till 0942hrs on the same day. I also noted that the hosts MY.NET.7.140 (326 times) and MY.NET.111.155 (282 times) were targeted the most times. The other hosts averaged about 60~70 alerts each.

This host could be running a vulnerability scan against the machines in your network, specifically looking for vulnerable IIS web servers susceptible to the Unicode attack(s). A check with *Dshield* and *Sam Spade* did not reveal any information about this IP address. I would recommend:

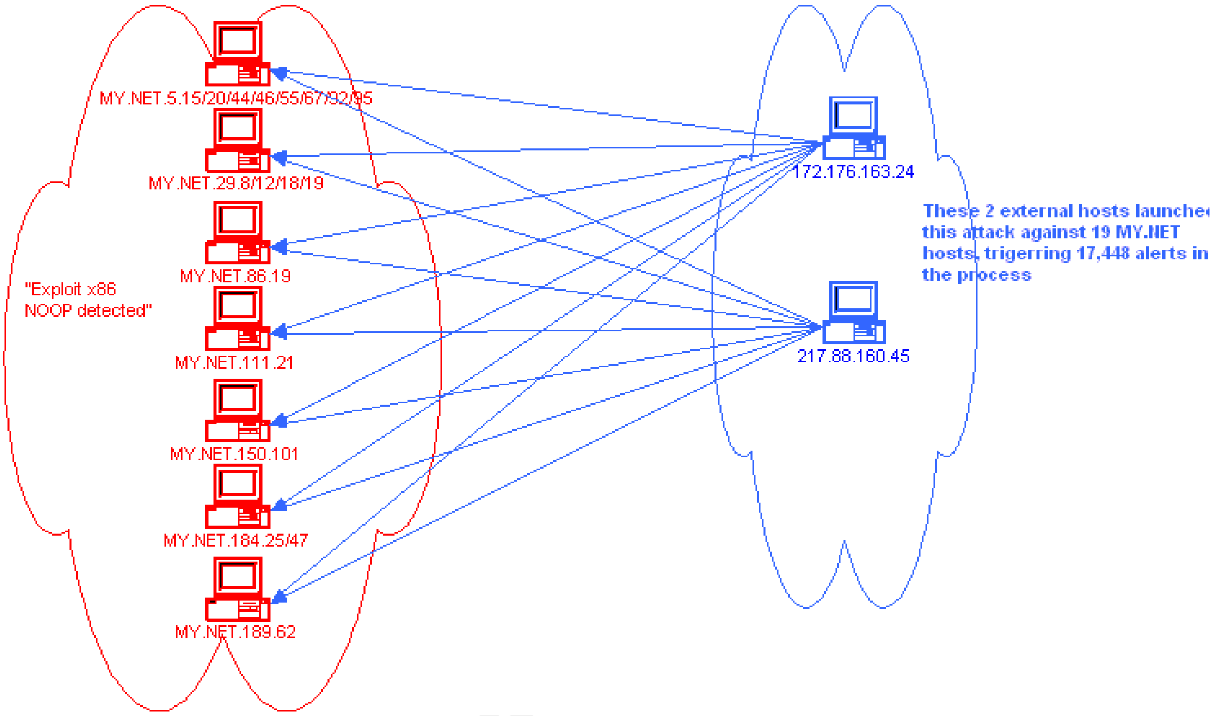
- Checking the raw logs to see there were any return traffic from your hosts to this IP, especially from MY.NET.7.140 and MY.NET.111.155. This might indicate a successful compromise.
- Ensure that the IIS web servers in your network are patched to the latest

versions.

- Block this IP from entering your network if it was confirmed that an attack was carried out.

Case 3 – Attacks involving the Exploit x86 NOOP

Reference to Table 6, the hosts at 172.176.163.24 and 217.88.160.45 featured prominently in terms of the Exploit x86 NOOP attacks. Upon further examination, I found that the targeted MY.NET hosts by these 2 IPs were similar:




**Link Graph 3**

A run of whois against these two addresses revealed them to be from America On Line and Deutsche Telekom respectively. This alert is also susceptible to false positives, such as traffic carrying binary data, jpg, GIF and bitmaps. I found out that all the destination ports for the alerts was port 80 (HTTP). This could indicate return HTTP traffic to web servers in MY.NET, whose payload contained NOOP bytes. I deduced that 172.176.163.24 and 217.88.160.45 belonged to Internet users who were accessing web servers in your network, but unintentionally triggered the NOOP alerts. Maybe they were transferring executables, pictures that contained NOOP bytes.

Reference to Table 4 and 5, I found that only incoming traffic triggered these alerts. In summary,

| | | |
|---|---|---|
| 1 | No. of alerts | 32,702 |
| 2 | No. of alerts where source port eq 80 | 702 |
| | - No. of unique MY.NET hosts targeted | 74 |
| 3 | No. of alerts where destination port eq 80 | 26,656 |
| | - No. of unique MY.NET hosts targeted | 21 |
| 4 | No. of alerts where destination port eq 119 | 5,521 |

```
   - No. of unique MY.NET hosts targected                       2
```

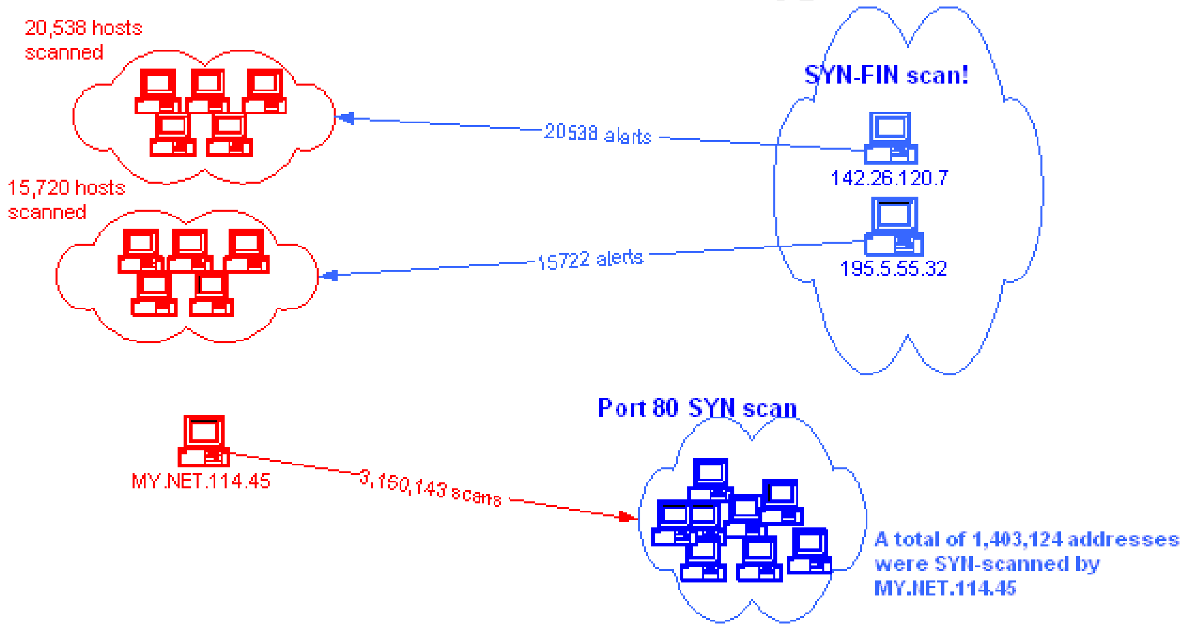
Source port of 80 indicated traffic from external web servers to MY.NET hosts. 74 hosts in MY.NET were involved, most likely belonging to users who were accessing to external web sites.

Destination port of 80 was explained earlier. I would recommend verifying whether the MY.NET hosts (as in Link Graph 3) are indeed legitimate web servers, and if possible, run vulnerability scans against them.

Destination port of 119 indicated postings to news servers in MY.NET. 2 were found, namely MY.NET.24.8 and MY.NET.81.42, and I would advise verifying these 2 machines are indeed running as news servers.

From the observations, I recommend disabling the “Exploit x86 NOOP” signature in the Snort IDS, which would reduce the amount of alerts due to false positives.

<u>Case 4 – Port scans activities against and from MY.NET hosts</u>




**Link Graph 4**

Two activities brought about concerns here. Firstly, your network was actively targeted by SYN-FIN scans from 2 particular external hosts. Secondly, MY.NET.114.45 sent out a barrage of SYN packets against port 80 of a wide range of external addresses. I suspect an automated portscan tool was run on this particular machine. I recommend checking on this machine for any possible compromise.

## 7. List of Detects

I referred to the SANS/FBI Top 20 List of the Most Critical Internet Security Vulnerabilities at http://www.sans.org/top20/[25], and grouped the attacks found in your network according to the categories of vulnerabilities. The remaining attacks were highlighted at the end of the section.

SANS/FBI Top Vulnerabilities to Windows Systems

| **W1 – Internet Information Services (IIS)** | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 98,201 | IIS Unicode attack | Unicode, which are basically hex-encoded characters, are used to attack IIS servers that are not able to handle improperly formatted HTTP requests. A remote attacker would be able to execute arbitrary commands on the server, such as *cmd.exe* as part of crafted HTTP request.<br><br>The alerts triggered could also be false positives, as Unicode characters could exist in legitimate web traffic. This was highlighted in Case 2 of the Link Graphs section. |
| 8,995 | CGI Null Byte | When the Snort http pre-processor detects a %00 in a http request, it will alert with "CGI Null Byte Attack". Attackers could use this as means to have arbitrary access to a web server.<br><br>However, the alerts could be false positives, instances where access to sites that use cookies with URL-encoded binary data, or when SSL encrypted is picked up. Having a packet dump is the only way to verify whether we have a real attack in our hands.<br><br>A couple of reports on this alert has concluded that such alerts were false alarms:<br><br>http://www.giac.org/practical/Joe_Ellis_GCIA.doc[26]<br>*(Joe Ellis' GCIA Practical)*<br>http://www.lurhq.com/idsindepth.html[27]<br>*(Johnny Calhoun's GCIA Practical)* |
| 2,393 | IDS552/web_iis ISAPI overflow ida nosize<br><br>&<br><br>IDS552/web_iis ISAPI overflow IDA internal | Many ISAPI extensions are vulnerable to buffer overflows. This event indicates a remote attacker has attempted to exploit a vulnerability in Microsoft IIS:<br><br>http://www.whitehats.com/info/IDS552[28]<br><br><br>An unchecked buffer in the Microsoft IIS Index Server ISAPI Extension could enable a remote intruder to gain SYSTEM access to the server:<br><br>http://www.eeye.com/html/Research/Advisories/AD2001 |

| | | |
|---|---|---|
| | | 0618.html[29]<br><br>The hosts MY.NET.97.24 and MY.NET.97.205 were triggering such alerts. These 2 hosts could be compromised. |
| 405 | NIMDA – Attempt to execute cmd or root from campus host | These attacks target the buffer overflow vulnerability of ISAPI extensions. Such attacks could cause Denial of Service or allow the execution of arbitrary code on the Web server.<br><br>Six MY.NET hosts were found generating these alerts:<br>- MY.NET.30.86<br>- MY.NET.97.24<br>- MY.NET.97.176<br>- MY.NET.114.15<br>- MY.NET.184.25<br>- MY.NET.97.205 |
| **Defensive Recommendations** | | |
| The SANS/FBI Top 20 site suggested a few approaches to protect against IIS attacks, such as applying the latest patches, eliminating sample applications and unmapping unnecessary ISAPI extensions.<br><br>I would also like to recommend checking the MY.NET hosts indicated earlier for any signs of compromise. Particularly MY.NET.97.24 and MY.NET.97.205, both featured as source of ISAPI and Nimda alerts. | | |

| **W4 NETBIOS – Unprotected Windows Network Shares** | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 70,614 | SMB Name Wildcard | Attackers could craft packets directed at port 137 to extract useful NetBIOS information like workstation name, domain and users currently logged in (NetBIOS Name Table Retrieval Query). Such packets would trigger the SMB Name Wildcard alerts.<br><br>There was a high number of alerts coming from source address 169.254.45.176. This address belonged to the Linklocal address space, as defined in RFC3330:<br><br>http://www.rfc-editor.org/rfc/rfc3330.txt[30]<br><br>Linklocal addresses are assigned to network interfaces when a local DHCP server is not available. The fact that none of the MY.NET hosts was the source of this attack, I concluded that the alerts (from source 169.254.45.176) originated from MY.NET hosts running Microsoft Windows 98. Refer to:<br><br>http://www.sans.org/y2k/072500-1200.htm[31] |
| 139 | SMB C access | This signature captures attempts to access the default administrative share C$. If successful, the attacker would be able to access the c: filesystem.<br><br>http://www.whitehats.com/info/IDS339[32] |

| | | |
|---|---|---|
| **Defensive Recommendations** | | |
| Some of the recommendations from the SANS/FBI Top 20 site:<br>- Disable Windows file sharing if not required, otherwise, enforce authenticated shares<br>- Deny sharing with hosts on the Internet<br>- Block ports used for Windows shares at your network perimeter i.e. 137-139 TCP and UDP, and 445 TCP and UDP. | | |

| **W5 Anonymous Logon – NULL sessions** | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 47 | NetBIOS NT Null session | This signature detects a Windows NT login as Nobody (nt-netbios-nullsession). Null sessions are used to list shares and users on a Windows NT server or client workstation.<br><br>http://www.whitehats.com/info/IDS204[33] |
| **Defensive Recommendations** | | |
| The recommendations pointed out in W4 could be adopted here. | | |

## SANS/FBI Top Vulnerabilities to Unix Systems

| **U1 Remote Procedure Calls (RPC)** | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 989 | External RPC call | The external RPC calls targeted port 111, the portmapper service. The attackers hoped to find out what services are running on a particular host and on which port the services are run. A total of 6 external hosts targeted 590 machines in your network:<br>- 216.101.67.45<br>- 211.168.183.66<br>- 67.34.61.114<br>- 67.32.137.235<br>- 210.251.104.22<br>- 143.225.151.30 |
| 167 | SUNRPC highport access<br><br>&<br><br>Attempted Sun RPC high port access | The signature used to detect this attack was based on access to port 32771 TCP or UDP. Hence, it was highly probable that these were false positives.<br><br>It would be a concern if those hosts that carried out the "external RPC call" were featured here, but fortunately, they did not. If they did, that meant that they got a response from the targeted host on which services were running, and attempted to connect to them. |
| **Defensive Recommendations** | | |
| Some recommendations which I wish to highlight:<br>- turn off any RPC service on your Unix hosts unless absolutely required<br>- install the latest patches if RPC services count not be removed<br>- block portmapper port 111 at your network perimeter<br>- block RPC "loopback" ports from 32770 to 32789 (TCP and UDP) | | |

| |
|---|
| |

| U4 Simple Network Management Protocol (SNMP) | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 156 | SNMP public access | This Snort signature attempts to detect any access to port 161 (SNMP) with the "public" community string.<br><br>The 150 occurrences was attributed to a one-sided traffic from 134.192.86.65 to MY.NET.190.13. The culprit resolved to a host in your network, according to *whois*.<br><br>I would recommend verifying whether the traffic was legitimate or due to some mis-configuration of equipment. |
| **Defensive Recommendations** | | |
| If SNMP is not required, it should be turned off. Otherwise, replace the default "public" community string. | | |

| U5 File Transfer Protocol (FTP) | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 1,283 | FTP passwd attempt | This signature detects any attempt to retrieve the passwd file from an FTP server.<br><br>The external host 218.19.12.57 triggered the alert against 80 MY.NET hosts. This address originated from China. |
| 16 | FTP DOS ftpd globbing | This signature detects any attempt to crash the ftpd server software by sending a wildcard request to create a DOS on vulnerable FTP servers:<br><br>http://www.whitehats.com/info/IDS487[34]<br><br>The 16 occurrences were attributed to a one-sided traffic from 213.133.108.15 to MY.NET.24.27. The culprit resolved to a host in Germany, according to *whois*. |
| **Defensive Recommendations** | | |
| The following recommendations are suggested:<br>- upgrade to latest version of FTP<br>- implement restrictive file permissions on the FTP server | | |

| U7 Line Printer Daemon (LPD) | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 1,973 | Connect to 515 from inside | The LPD daemon listens on TCP port 515. Many implementations of LPD contain programming flaws which led to buffer overflow situations, allowing attackers to run arbitrary code with root privileges.<br><br>The hosts MY.NET.97.20, MY.NET.97.79, MY.NET.97.93 |

| | | |
|---|---|---|
| | | and MY.NET.97.122 could be running a portscan on 132.250.182.61. This was characterized by the incrementing (by one every attempt) source port used. |
| 1,384 | Connect to 515 from outside | The 1,384 occurrences were attributed to incoming traffic from 131.118.229.7 to MY.NET.24.15. |
| **Defensive Recommendations** | | |
| The following recommendations are suggested:<br>- conduct a check on MY.NET.97.20, MY.NET.97.79, MY.NET.97.93 and MY.NET.97.122. These machines could be compromised.<br>- verify whether the traffic between 131.118.229.7 and MY.NET.24.15 is legitimate. If the machine does not need to act as a print server for remote requests, then the LPD service should be blocked. | | |

| **U9 BIND/DNS** | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 3,167,350 | Possible scans for BIND weaknesses | As observed in the scans logs, both hosts MY.NET.1.3 and MY.NET.1.4 were sending out UDP packets destined to port 53 (DNS) of 76,316 external hosts. |
| **Defensive Recommendations** | | |
| I would recommend checking on the hosts MY.NET.1.3 and MY.NET.1.4 for any signs of compromise. The fact that port 53 is related to BIND/DNS servers, and the large number of hosts targeted suggested a possibility that these hosts were looking for weak hosts to compromise. | | |

I also highlighted the other types of attacks:

| **Possible reconnaissance attempts** | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 36,271 | SYN-FIN scans | Packets with SYN-FIN flags set do not occur naturally and indicates an intentional probe. It is probably a single packet OS detection probe:<br><br>http://www.whitehats.com/info/IDS198[35]<br><br>The SYN-FIN scans against your network were targeted at port 21 (FTP). The attackers were most likely looking for vulnerable FTP servers, such as WU-FTPD. Similar sightings of SYN-FIN scan to port 21 were found:<br><br>http://www.dshield.org/pipermail/list/2003-July/009146.php (*James C. Slora, Jr*)[36]<br>http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00446.html (*Dave R*)[37]<br>http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00111.html (*Al Williams*)[38] |

| | | |
|---|---|---|
| 8,861 | Queso fingerprint | The Queso fingerprinting tool is used to determine the OS running on the targeted servers. Queso packets are characterized by the SYN, ECN and CWR (Reserved bits 1 & 2) flags set to 1, and a high TTL value. However, legitimate traffic might also bear the same characteristics, such as traffic across ECN-enabled routers.<br><br>Toby discussed the impact of the use of ECN/CWR bits (RFC 3168 previously RFC 2481) for network QoS on Intrusion Detection:<br><br>http://www.securityfocus.com/infocus/1205[39]<br><br>The IP 213.186.35.9 seemed to be probing your network for listening ports on<br>- ports 80, 81, 3128, 6588, 8080, 8081, 8000, 8001 (proxy related)<br>- port 23 (telnet)<br><br>The traffic was characteristic of a portscan in action, such as incrementing source port with every probe. |
| 2,402 | Null scans | Packets from a Null scan attempt are characterized by zero-ed TCP SEQ and ACK numbers, and all TCP flags. Null scans are used to detect the open ports on the targeted servers by observing the responses.<br><br>The top source of the NULL scan originated from 213.176.8.2. This IP was also culprit of launching the NMAP fingerprint and SYN-FIN against the hosts in your network. |
| 726 | Nmap TCP ping | TCP packets with the ACK number of zero and ACK flag set would trigger this alert.<br><br>http://www.whitehats.com/info/IDS28[40]<br><br>The remote attack could be using Nmap to probe the servers in your network. |
| 21 | Probable Nmap fingerprint attempt | A remote attacker used nmap to fingerprint the OS running on your servers. The packets are characterized by the SYN, FIN, URG and PUSH flags all set:<br><br>http://www.whitehats.com/info/IDS5[41] |

| **Defensive Recommendations** |
|---|
| The reconnaissance attempts highlighted used packets that are out-of-spec (not likely to occur in normal traffic) to solicit response from servers, such as determining the OS running or which ports were opened. All the probe attempts originated from external hosts (fortunately). A stateful firewall would be able to block these out-of-spec packets from entering your network. |

<table>
<tr><th colspan="3">Shellcode attacks</th></tr>
<tr><th>Alert Count</th><th>Attack</th><th>Description</th></tr>
<tr><td>32,702</td><td>Exploit x86 NOOP</td><td rowspan="5">Shellcode exploits make use of platform specific operations (such as the 0x90 character which represents NOOP in x86 machine code) to hide the buffer overflow attempts. Hence, the IDS signatures detect such attacks by inspecting the packet payload for specific strings of hex bytes.<br><br>Exploit x86 NOOP:<br>http://www.whitehats.com/info/IDS181[42]<br><br>Exploit x86 stealth NOOP (using the jmp 0x02):<br>http://www.whitehats.com/info/IDS291[43]<br><br>Exploit x86 setuid 0 (using the setuid(0) system call for x86 platform):<br>http://www.whitehats.com/info/IDS283[44]<br><br>Exploit x86 setgid 0 (using the setgid(0) system call for x86 platform):<br>http://www.whitehats.com/info/IDS284[45]<br><br>The above alerts are susceptible to false positives because the very signatures used to detect them occur in normal legitimate network traffic as well. For example, the byte strings may occur in binary files downloads. Another example (x86 Exploit NOOP – false alarm):<br><br>http://www.giac.org/practical/David_Oborn_GCIA.html[46]<br>http://cert.uni-stuttgart.de/archive/intrusions/2003/05/msg00090.html[47]<br><br>The Exploit NTPDX buffer overflow attack attacks vulnerable implementations of ntpd and xntpd daemons:<br>http://www.whitehats.com/info/IDS492[48]<br><br>The signature works be detecting any UDP packets destined for port 123 with the length greater than 128 bytes. According to the URL above, there was mention of some unusual implementations or obscure options that might cause longer packets than normal to be sent.</td></tr>
<tr><td>64</td><td>Exploit x86 stealth NOOP</td></tr>
<tr><td>49</td><td>Exploit x86 setuid 0</td></tr>
<tr><td>34</td><td>Exploit x86 setgid 0</td></tr>
<tr><td>25</td><td>Exploit NTPDX buffer overflow</td></tr>
<tr><th colspan="3">Defensive Recommendations</th></tr>
<tr><td colspan="3">It would be more difficult to block such attacks than detecting them because more often than not, normal network traffic would be the culprit. The only way to verify whether such attacks did occur is by inspecting the raw packet logs.</td></tr>
</table>

| **Worm activity** | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 133,659 | High port tcp 65535 – possible Red Worm traffic | The Snort signature appeared to be locally created and not found in a standard Snort rulebase, possibly:<br><br>alert tcp any any -> any 65535 (msg: "High port 65535 tcp – possible Red Worm – traffic"; )<br><br>alert udp any any -> any 65535 (msg: "High port 65535 udp – possible Red Worm – traffic"; )<br><br>The Red Worm is also known as the Adore Worm. The signature was probably created to detect the Adore Worm, which spreads in Linux via vulnerabilities found in BIND named, wu-ftpd, rcp.statd and lpd services. A compromised host opens a backdoor in the port 65535. The signature triggered an alert as long as either the source or destination TCP port equals 65535. However, port 65535 could also exist in a legitimate TCP connection, giving rise to false positives.<br><br>http://www.sans.org/y2k/adore.htm[49]<br>http://www.dials.ru/english/inf/linux_adore.htm[50]<br>http://www.f-secure.com/v-descs/adore.shtml[51]<br>http://www.giac.org/practical/Michael_Reiter_GCIH.zip[52]<br><br>William Stearns has written a script to detect the presence of the Adore Worm called *adorefind*, which could be downloaded from:<br><br>http://www.ists.dartmouth.edu/IRIA/knowledge_base/tools/adorefind.htm[53] |
| 1,311 | High port udp 65535 – possible Red Worm traffic | |
| 50 | MYPARTY – possible My Party infection | MyParty is a mass-mailing email worm. Part of the worm code opens a backdoor Trojan that allows a remote attacker to control the compromised host:<br><br>http://securityresponse.symantec.com/avcenter/venc/data/w32.myparty@mm.html[54]<br><br>According to the description, the backdoor Trojan contacts the web site at 209.151.250.170 and execute instructions based on the contents of the site. Matt Yackley captured some of the web activity from the Trojan at:<br><br>http://www.incidents.org/archives/intrusions/msg03040.html[55] |
| **Defensive recommendations** | | |
| Perhaps the signature used to detect the Red Worm traffic was too | | |

| |
|---|
| simplistic, resulting in false positives. Rather, I recommend running the *adorefind* script to detect the presence of this worm in your network.<br><br>As for the MyParty worm infection, I would recommend checking the firewall logs for any outgoing connection to 209.151.250.170. If there are, then the affected machines should be checked for the presence of the file msstask.exe. |

**Trojan activity**

| **Alert Count** | **Attack** | **Description** |
|---|---|---|
| 527 | Possible Trojan server activity | The signature for this alert was probably crafted to detect the TCP port 27374. Port 27374 is related to a variety of Trojans, such as the SubSeven Trojan.<br><br>Simple as it was, the signature might attribute to false positives, because port 27374 could be the ephemeral port used in a legitimate transaction. |
| 49 | RFB – possible WinVNC | WinVNC is the Microsoft Windows version of AT&T's VNC (Virtual Network Computing). VNC is a remote control software that allows a user to view and interact with one computer using another computer anywhere in the Internet.<br><br>http://home.earthlink.net/~jknapka/vncpatch.html[56]<br><br>A default installation of VNC serves out the Java applet via port 5800. Ports 5900-5903 is used to serve the RFB (remote frame buffer) sessions between the client and server. The signature for this alert was probably crafted to detect the presence of any WinVNC sessions by looking for traffic where ports 5900-5903 was used.<br><br>The signature is susceptible to false positives because ports 5900-5903, being above 1024, might be used is normal traffic. |
| 39 | DDOS shaft client to handler | shaft is a DDOS tool that is made up of a few handlers and a large number of agents. The attacker uses telnet to communicate with the handlers. A detailed analysis of the shaft DDOS tool can be found at:<br><br>http://home.adelphi.edu/~spock/shaft_analysis.txt[57]<br><br>Traffic flow between the client and handlers is characterized by the use of tcp port 20432. A sample Snort signature can be found at:<br><br>http://www.whitehats.com/info/IDS254[58]<br><br>The signature is susceptible to false positives because port 20432 might be used is normal |

| | | |
|---|---|---|
| | | traffic. |
| 8 | DDOS mstream handler to client | mstream is another DDOS tool that consists of a handler and an agent component. The agent performs the actual DDOS attack, whereas the handler issues commands to the agent to begin the attack. Both components were designed to run on Unix systems.<br><br>http://www.cert.org/incident_notes/IN-2000-05.html[59]<br>http://staff.washington.edu/dittrich/misc/mstream.analysis.txt[60]<br><br>The signature for detecting mstream handler to client traffic was based on the use of source port 15104/tcp or 12754/tcp. Likewise for mstream client to handler, the use of destination port of 15104/tcp or 12754/tcp would trigger the alert.<br><br>This signature is susceptible to false positives because ports 15104 and 12754 might appear in normal traffic. There was no cause for concern here because the MY.NET hosts that appeared in the handler->client alerts did not appear in the client->handler alerts logs. If the tool did exist, then there would be two-way communications |
| 3 | DDOS mstream client to handler | |
| 7 | Back Orifice | Back Orifice is a Trojan tool that allows an attacker to take over control of another computer. There was reported Back Orifice activity on MY.NET hosts:<br><br>- MY.NET.153.113<br>- MY.NET.150.21<br>- MY.NET.114.88 |
| **Defensive recommendations** | | |
| Most of the signatures used to detect the Trojans are based on the identification of the ports used in the traffic flow. Hence, they are susceptible to false positives. An inspection of the packet payload would give a clearer picture as to whether a Trojan activity took place.<br><br>I recommend checking on the hosts MY.NET.153.113, MY.NET.150.21 and MY.NET.114.88 for any signs of Back Orifice. | | |

| **Unusual network traffic** | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 1,530 | TCP SRC and DST outside network | These alerts are result of spoofed traffic that originated from your network. Packets with the source and destination addresses outside your address space should never be seen in normal traffic. |
| 10 | ICMP SRC and DST outside network | |
| 332 | Incomplete packet fragments | A search of this alert produced the following information: |

| | | |
|---|---|---|
| | discarded | *This message is generated by the Snort defragmentation preprocessor when packets bigger than 8Kbytes that are more than half empty when the last fragment is received are discarded.*<br><br>http://archives.neohapsis.com/archives/snort/2001-02/0320.html[61]<br><br>Possible causes of such conditions are transmission errors, broken stacks or fragmentation attacks (evasion?). It was noted that the source and destination ports of the all these packets were 0. |
| 276 | Tiny fragments – possible hostile activity | Older versions of Snort contained the *minfrag* preprocessor. The *minfrag* preprocessor checks for fragmented packets. If the packet is a fragment, and its size is less than the threshold value set, then the alert will trigger.<br><br>For example, to generate an alert each time a packet fragment less than 128bytes in size is received:<br><br>preprocessor: minfrag 128 any<br><br>A majority of the alerts was triggered by the external host 208.180.168.58. |
| **Defensive recommendations** | | |
| In order to prevent spoofed packets from exiting your network, I recommend implementing egress filtering at the perimeter router. | | |

| **Vulnerable services – TFTP** | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 118 | TFTP – internal TCP connection to external tftp server | TFTP (Trivial File Transfer Protocol) is a simple protocol used to transfer files. The protocol is defined in RFC 1350:<br><br>http://www.rfc-editor.org/rfc/rfc1350.txt[62]<br><br>The weakness of the protocol is that no authentication is required (the fact that it is trivial to begin with). The protocol was also used by Worms to download code from another location. The TFTP server listens on port 69 TCP and UDP. |
| 64 | TFTP – internal UDP connection to external TFTP server | |
| 22 | TFTP – external TCP connection to internal tftp server | |
| 6 | TFTP – external UDP connection to internal TFTP server | |

| **Defensive recommendations** |
|---|
| I would recommend a re-evaluation of the availability of TFTP services in your network to external hosts. And also whether outgoing TFTP connections can be blocked altogether. TFTP has often being linked to worms and Trojans because of its simplicity and no authentication required. |

| **IRC-related attacks** | | |
|---|---|---|
| **Alert Count** | **Attack** | **Description** |
| 16 | IRC evil – running XDCC | I managed to locate a link to the operation of the IRC XDCC bot from Sanjay Menon's GCIA practical:<br><br>http://security.duke.edu/cleaning/xdcc.html[63]<br><br>The XDDC backdoor allows a remote attacker to take over a compromised host.<br><br>MY.NET hosts that could possibly be compromised were:<br>- MY.NET.82.36<br>- MY.NET.80.209<br>- MY.NET.74.216<br>- MY.NET.198.221 |
| 3 | Possible incoming XDCC send | |
| 16 | Possible sdbot floodnet | Sdbot is a backdoor Trojan that allows an attacker to unauthorized access to an infected computer:<br><br>http://securityresponse.symantec.com/avcenter/venc/data/backdoor.sdbot.html[64]<br><br>The Trojan connects to an IRC server, joins a specific channel, and notifies the attacker by sending a private message. The Trojan then awaits commands from the attackers via IRC.<br><br>12 MY.NET hosts were detected, most from the MY.NET.97.x segment:<br>- MY.NET.150.85 and MY.NET.150.121<br>- MY.NET.153.111<br>- MY.NET.97.10/16/18/68/74/100/124/184<br>- MY.NET.98.15<br><br>I also noticed that the destination of these hosts was port 6667 of 213.186.35.9, the port used for IRC. This external IP was also observed conducting a portscan of your network (refer to the Queso fingerprint attack). |
| 3 | User joining warez channel | I suspect this was a locally created signature to detect any users joining a warez channel via IRC. |

| **Defensive recommendations** |
|---|
| Communications using IRC channels is one of the ways Trojans employ. Hence, the use of Snort signatures to detect the strings in IRC communications related to possible compromise or attacks.<br><br>I would recommend a check on the MY.NET hosts involved in the attacks for |

```
any signs of compromise.
```

## 8. Registration information of 5 external addresses

The following addresses were selected because of their involvement in possible attacks against your network:

**#1 – 24.84.205.243**

**Why?** – There was a two way communication between this host and MY.NET.82.36, which generated close to 132,468 Red Worm alerts (refer to Link Graphs – Case 1):

```
$ whois -h whois.arin.net 24.84.205.243

OrgName:    Shaw Communications Inc.
<< snipped >>

NetRange:   24.80.0.0 - 24.87.255.255
CIDR:       24.80.0.0/13
NetName:    SHAW-COMM
<< snipped >>

<< snipped >>

# ARIN WHOIS database, last updated 2003-09-30 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

A search of Google.com revealed Shaw Comms as a Canadian-based communications company whose core business is providing broadband cable TV, Internet and satellite services. Loading the IP in InfoBear.Com's NSLookup page, the following information was retrieved:

```
Output of:
nslookup -q=A 24.84.205.243 ns1.worldnet.att.net

Server: ns1.worldnet.att.net
Address: 204.127.129.1

Name: h24-84-205-243.vc.shawcable.net
Address: 24.84.205.243
```

The hostname suggested the IP belonged to an ISP subscriber. A check on Dshield did not return any hits on this address.

**#2 – 80.204.44.179**

**Why?** – This IP was suspected of running some vulnerability scans against the servers in your network (refer to Link Graphs – Case 2):

```
$ whois -h whois.ripe.net 80.204.44.179
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      80.204.44.176 - 80.204.44.183
```

```
netname:       NORLIGHT--SRL
descr:         NORLIGHT  SRL
country:       IT
<< snipped >>
notify:        network@cgi.interbusiness.it
changed:       network@cgi.interbusiness.it 20020214
source:        RIPE
<< snipped >>
changed:       datacomnet@telecomitalia.it 20011212
source:        RIPE

person:        Luca Camarda
address:       NORLIGHT  SRL
address:       V.CELLINI 8
address:       I- 21100  CASSANO MAGNAGO (VA)
address:       Italy
<< snipped >>
changed:       domain@cgi.interbusiness.it 20020214
source:        RIPE
```

The host originated from Italy. A search for "Norlight Italy" returned an Italian-based company specializing in lighting solutions. However, the domain lookup of www.norlight.it did not match the IP net range of the earlier whois result. From Infobear.com's NSLookup page, the information returned was:

```
Output of:
nslookup -q=A 80.204.44.179 ns1.worldnet.att.net
Server: ns1.worldnet.att.net
Address: 204.127.129.1

Name: host179-44.pool80204.interbusiness.it
Address: 80.204.44.179
```

The domain *interbusiness.it* belonged to Telecom Italia, an Internet service provider. The hostname suggested the IP belonged to an ISP subscriber. A check on Dshield did not return any hits on this address.

**#3 – 142.26.120.7**

**Why?** – This host carried out 20,538 SYN-FIN scans against 20,538 machines in your network.

```
$ whois -h whois.arin.net 142.26.120.7

OrgName:    British Columbia Systems Corporation
OrgID:      BCSC
<< snipped >>
Country:    CA

NetRange:   142.26.0.0 - 142.26.255.255
CIDR:       142.26.0.0/16
NetName:    BCSYSTEMS5
NetHandle:  NET-142-26-0-0-1
Parent:     NET-142-0-0-0-0
NetType:    Direct Assignment
NameServer: DNS.GOV.BC.CA
NameServer: DNS1.GOV.BC.CA
NameServer: DNS2.GOV.BC.CA
```

```
NameServer: DNS3.GOV.BC.CA
<< snipped >>

# ARIN WHOIS database, last updated 2003-09-30 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

The domain suggested that the host originated from the Canadian Government's Class B address space. A check on Dshield did not return any reports on this address.

**#4 – 195.5.55.32**

**Why?** – This host garnered a total of 15,722 SYN-FIN scans against 15,720 hosts in your network:

```
$ whois -h whois.ripe.net 195.5.55.32
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      195.5.0.0 - 195.5.63.255
netname:      UA-UKRTELECOM-970717
descr:        Provider Local Registry
descr:        PROVIDER
country:      UA
<< snipped >>
```

UKRTelecom is an Internet Service Provider in Ukraine. A check on Dshield did not return any reports on this address.

**#5 – 213.176.8.2**

**Why?** – IP 213.176.8.2 was responsible for conducting a mixture of NULL, NMAP fingerprint and SYN-FIN scans against your network:

```
$whois -h whois.ripe.net 213.176.8.2
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      213.176.8.0 - 213.176.8.255
netname:      AKU
descr:        Amir Kabir University of Technology
descr:        Tehran
country:      IR
<< snipped >>

person:       Saied Mohammad Taghi Lavasani
address:      Computer and Information Center
address:      Amir Kabir University of Technology
address:      Hafez Ave. No 424
address:      Tehran
address:      Iran
<< snipped >>
```

This IP originated from a host within the Class C address space of a University in Tehran, Iran. From Infobear.com's NSLookup:

```
Output of:
nslookup -q=A 213.176.8.2 ns1.worldnet.att.net

Server: ns1.worldnet.att.net
Address: 204.127.129.1

Name: cic.aut.ac.ir
Address: 213.176.8.2
```

I could not make whether the hostname represented a client or a server machine. No reports of this IP was found in Dshield.

## 9. Conclusions and recommendations

I have so far taken a look at the alerts that were generated from your network within a short span of 5 days. The figures would seemed alarming at first, but upon analysis of the alerts, I singled out those that required attention, those that required further analysis and those that were possibly due to false alarms.

**General observations**

Generic signatures – There were instances where the alert signature was too generic, possibly creating a large number of alerts. For example, the Red Worm detection signature might mistake legitimate web traffic for attacks. I would advise the alert signatures be more clearly defined, such as looking at the payload content. In the case of the Red Worm, the *Adorefind* utility could be deployed instead to detect the existence of the worm, rather than flooding the alert logs with false positives.

There were a couple of locally created signatures to detect specific host activities. If the intention was so, then I would recommend filtering these alerts or employ other methods of host activity logging, such as Web server logs.

False positives – There were several instances of alerts which were susceptible to false positives. For example, Unicode, x86 NOOP, CGI Null Byte and Queso. The detection of some of these attacks could be disabled in Snort, if required.

**Possible compromised MY.NET hosts**

**MY.NET.82.36** – This machine could possibly be compromised. The events presented in the section Link Graphs – Case 1 justified a check on this machine for any signs of compromise.

**MY.NET.114.45** – This machine was carrying out portscans against a wide range of external addresses. Refer to Link Graphs – Case 4.

**MY.NET.97.24 and MY.NET.97.205** – These two hosts could possibly be compromised. They were detected running NIMDA-related attacks against external hosts.

**MY.NET.1.3 and MY.NET.1.4** – These two could be running BIND-related attacks against external machines.

**Network targeting**

SYN-FIN probes – There was a significant amount of SYN-FIN probes against your network, mostly originating from 142.26.120.7 and 195.5.55.32. Your perimeter firewall should be able to block such scans from reaching the internal machines, but it would be good to verify. Such probes were possibly crafted using automated tools.

Unicode attacks – There was noticeable activity from 80.204.44.179, triggering 26,397 alerts in the process that involved 437 MY.NET hosts. I recommend a vulnerability scan to be conducted against the web servers in your network, and ensure that they were patched. The signs possibly indicated a successful attack against MY.NET.7.140 and MY.NET.111.155.

Open proxy scans – The host 213.186.35.9 was detected probing for open proxies in your network. I recommend patching the proxy servers in your network, if any, and implement access lists to restrict access to internal hosts only. There was also activity from some MY.NET hosts to the IRC port of this address.

Multiple fingerprint attempts was detected from 213.176.8.2. As with SYN-FIN scans, your perimeter firewall should be able to block such attempts.

**Unsafe services**

NETBIOS – such traffic should be filtered at your network's perimeter routers.

RPC – turn these services off unless absolutely required.

FTP – upgrade to the latest version/patch/build of the FTP server.

LPD – evaluate the need for serving print services to external hosts.

TFTP – block the serving of TFTP services to external hosts.

Back Orifice – Noted BO traffic on hosts MY.NET.153.113, MY.NET.150.21 and MY.NET.114.88.

# Annex A: Pre-Analysis stage for Part 3: Tools used and procedures

## Task: Concatenation of various alert, scans and OOS files

```
$ cp alert.030709 alert-all.txt
$ cat alert.030710 >> alert-all.txt
$ cat alert. 030711 >> alert-all.txt
$ cat alert. 030712 >> alert-all.txt
$ cat alert. 030713 >> alert-all.txt

$ cp scans.030709 scans-all.txt
$ cat scans.030710 >> scans-all.txt
$ cat scans.030711 >> scans-all.txt
$ cat scans.030712 >> scans-all.txt
$ cat scans.030713 >> scans-all.txt

$ cp OOS_Report_2003_07_09_2126 OOS_Report-all.txt
$ cat OOS_Report_2003_07_10_4402 >> OOS_Report-all.txt
$ cat OOS_Report_2003_07_11_27931>> OOS_Report-all.txt
$ cat OOS_Report_2003_07_12_20109 >> OOS_Report-all.txt
$ cat OOS_Report_2003_07_13_9896>> OOS_Report-all.txt
```

## Task: Separating alerts from portscan events

```
$ grep -v portscan alert-all.txt > alert-all.filtered.txt
```

## Task: Use the separator % for entry into MySQL database (learnt from Brandon Newport's GCIA paper[65]

**Alerts**

```
Step 1: sed -e 's/\[\*\*\]/%/g' alert-all.filtered.txt #replace [**] with %
Step 2: sed -e 's/->/%/g' input-file #replace -> directional arrow with %
Step 3: sed -e 's/decode:/decode-/g' input-file #the string "decode:" will
cause \ problems with replacement of : later
Step 4: sed -e 's/:/ % /4' input-file #replace 4th occurrence of : in input\
string, which separates the destination IP address and destination port
Step 5: sed -e 's/:/ % /3' input-file #replace 3rd occurrence of : in input\
string, which separates the source IP address and source port
```

**Overall command for alerts**

```
$ sed -e 's/\[\*\*\]/%/g' alert-all.filtered.txt | sed -e 's/->/%/g' | sed
-e 's/:/ % /4' | sed -e 's/:/ % /3' > alert-all.mysql

$ head -3 alert-all.mysql #sample output
07/09-00:00:02.463431  % CS WEBSERVER - external web traffic %
210.241.238.236 % 62639 % MY.NET.100.165 % 80
07/09-00:00:04.180310  % CS WEBSERVER - external web traffic %
210.241.238.236 % 62642 % MY.NET.100.165 % 80
07/09-00:00:04.578871  % MY.NET.30.4 activity % 66.196.72.70 % 53835 %
MY.NET.30.4 % 80
```

**Overall command for scans**

```
$ awk '{print $4,$6,$7}' scans-all.txt | sed -e 's/ / % /g' | sed -e 's/:/ % /2' | sed -e 's/:/ % /1' > scans-all.mysql
```

## Task: Enter the alerts information into MySQL database thanks to Brandon Newport's excellent paper again ☺

```
$ mysql -u zz -p part3
Password:
mysql> \c
mysql> create table alert
    -> ( date varchar (21),
    -> attack varchar (50),
    -> src varchar (15),
    -> srcp varchar (6),
    -> dst varchar (15),
    -> dstp varchar (6));
mysql> load data infile '/home/jwong/GCIA/alert-all.mysql' into table alert fields terminated by '%' ;
```

## Task: Alerts statistics (Table 1)

```
mysql> select attack,count(distinct src),count(distinct dst),count(*) as count from alert group by attack order by count desc;
```

## Task: Top Source and Destination, MY.NET and External, Top Talkers, Listeners

```
Top 10 Source from MY.NET (Table 2)
mysql> select src,count(distinct attack),count(*) as count from alert where src like "%MY.NET%" group by src order by count desc limit 10 ;

Top 10 Destination from MY.NET (Table 3)
mysql> select dst,count(distinct attack),count(*) as count from alert where dst like "%MY.NET%" group by dst order by count desc limit 10 ;

Types of attacks launched by MY.NET hosts (Table 4)
mysql> select attack,count(distinct src),count(*) as count from alert where src like "%MY.NET%" group by attack order by count desc ;

Types of attacks launched against MY.NET hosts (Table 5)
mysql> select attack,count(distinct dst),count(*) as count from alert where dst like "%MY.NET%" group by attack order by count desc ;

Top 10 Source from external (Table 6)
mysql> select src,count(distinct attack),count(*) as count from alert where src not like "%MY.NET%" group by src order by count desc limit 10 ;

Top 10 Destination from external (Table 7)
mysql> select dst,count(distinct attack),count(*) as count from alert where dst not like "%MY.NET%" group by dst order by count desc limit 10 ;

Top source/attack pair (Table 8)
mysql> select src,attack,count(*) as count from alert group by src,attack order by count desc limit 10 ;

Top source/attack pair (Table 9)
```

```
mysql> select dst,attack,count(*) as count from alert group by dst,attack
order by count desc limit 10 ;

Top Source/Destination Pair (Table 10)
mysql> select src,dst,count(*) as count from alert group by src,dst order
by count desc limit 10 ;
```

## Task: Reviewing the scans logs

```
$ wc –l scans-all.txt
12281498 scans-all.txt

The scans were successfully imported into a MySQL database.

mysql> \c
mysql> create table scans
    -> ( src varchar (15),
    -> srcp varchar (6),
    -> dst varchar (15),
    -> dstp varchar (6)
    -> type varchar (15));
mysql> load data infile '/home/jwong/GCIA/scans-all.mysql' into table scans
fields terminated by '%' ;

Top 5 Scans type (Table 11)
mysql> select type,count(distinct src),count(*) as count from scans group
by type order by count desc limit 5 ;
```

# List of References

[1] http://www.nwfusion.com/columnists/2002/0930tolly.html. There was also a discussion thread in Dshield.org on GoToMyPC at http://www.dshield.org/pipermail/list/2002-May/004014.php.

[2] Hompage of Expertcity's GoToMyPC. URL: http://www.gotomypc.com.

[3] These were extracted from the technical documents downloadable from the site.

[4] Homepage of TruSecure. URL: http://www.trusecure.com

[5] TruSecure's SiteSecure: URL: http://www.trusecure.com/solutions/assurance/sitesecure/

[6] AES – Advanced Encryption Standard, the algorithm selected by the National Institute of Standards and Technology (NIST) as the successor to DES (Data Encryption Standard).

[7] CFB – Cipher Feedback Mode.

[8] The ID is used to identify the PC and the access code (alphanumeric password) serves to authenticate user's access to the PC.

[9] The Hypertext Transfer Protocol version 1.0 – HTTP/1.0 (RFC1945) can be found at http://www.rfc-editor.org/rfc/rfc1945.txt

[10] Homepage of TotalRC.net. URL: http://www.totalrc.net.

[11] eBLVD remote can be downloaded from http://www.eblvd.com.

[12] An article on the use of open proxies for spamming http://zdnet.com.com/2100-1106-958847.html

[13] Murdoch, Don [incidents.org list] Proxy scan attempts. URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/04/msg00176.html

[14] Kibler, John R. [Dshield list] Scans on ports 3128, 8080 and 80. URL: http://www.dshield.org/pipermail/list/2002-May/004161.php

[15] I found a similar observation and response in Bergen, Keith. [Neohapsis archives] Logging of connects to port 6346. URL: http://archives.neohapsis.com/archives/incidents/2003-04/0063.html.

[16] Rohan, Amin. [incidents.org list] Strange NMAP from Port 80 to Linux Box. URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00027.html

[17] Brenton, Chris. [incidents.org list] Strange NMAP from Port 80 to Linux Box. URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00039.html

[18] Thomas, Ashley. [incidents.org list] LOGS; GIAC GCIA Version 3.3 Practical Detect. URL: http://cert.uni-stuttgart.de/archive/intrusions/2002/12/msg00167.html

[19] Description of the W32/SQLSlammer worm. URL: http://vil.nai.com/vil/content/v_99992.htm

[20] Analysis of the Port 1434 MS-SQL Worm. URL: http://isc.incidents.org/analysis.html?id=180

[21] A complete disassembly of the MS-SQL Slammer worm code. URL: http://www.nextgenss.com/advisories/mssql-udp.txt

[22] Fred Thiele's GCIA Practical. URL: http://www.giac.org/practical/GCIA/Fred_Thiele_GCIA.pdf

[23] SANS' Commonly Probed Ports. URL: http://www.sans.org/y2k/ports.htm

[24] The Snort FAQ. URL: http://www.snort.org/docs/FAQ.txt

[25] SANS' The Twenty Most Critical Internet Security Vulnerabilities. URL: http://www.snort.org/docs/FAQ.txt

[26] Joe Ellis' GCIA Practical. URL: http://www.giac.org/practical/Joe_Ellis_GCIA.doc

[27] Johnny Calhoun's GCIA Practical. URL: http://www.lurhq.com/idsindepth.html

[28] Whitehat's arachNIDS – The Intrusion Event Database. ***"IIS ISAPI OVERFLOW IDA"***. URL: http://www.whitehats.com/info/IDS552

[29] eEye Digital Security. ***"Microsoft Internet Information Services Remote Buffer Overflow (SYSTEM Level Access)"***. URL: http://www.eeye.com/html/Research/Advisories/AD20010618.html

[30] Special-use IPv4 addresses – RFC3330. URL: http://www.rfc-editor.org/rfc/rfc3330.txt

[31] Detect analyzed 7/25/00 – use of Linklocal addresses by Windows 98 workstations. URL: http://www.sans.org/y2k/072500-1200.htm

[32] Whitehat's arachNIDS – The Intrusion Event Database. ***"NETBIOS SMB-C$ACCESS"***. URL: http://www.whitehats.com/info/IDS339

[33] Whitehat's arachNIDS – The Intrusion Event Database. ***"NETBIOS NT-NULL-SESSION"***. URL: http://www.whitehats.com/info/IDS204

[34] Whitehat's arachNIDS – The Intrusion Event Database. ***"DOS-FTPD-GLOBBING"***. URL: http://www.whitehats.com/info/IDS487

[35] Whitehat's arachNIDS – The Intrusion Event Database. ***"SYN FIN SCAN"***. URL: http://www.whitehats.com/info/IDS198

[36] James C. Slora Jr. [Dshield list] Port 21 to Port 21 Scan. URL: http://www.dshield.org/pipermail/list/2003-July/009146.php

[37] Dave R. [Incidents.org list] LOGS GIAC GCIA Version 3.2 Practical Detect. URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00446.html

[38] Williams, Al. [incidents.org list] LOGS; GIAC GCIA Version 3.3 Practical Detect. URL: http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00111.html

[39] Miller, Toby. ECN and its impact on Intrusion Detection. URL: http://www.securityfocus.com/infocus/1205

[40] Whitehat's arachNIDS – The Intrusion Event Database. ***"PROBE-NMAP_TCP_PING"***. URL: http://www.whitehats.com/info/IDS28

[41] Whitehat's arachNIDS – The Intrusion Event Database. ***"PROBE-NMAP_FINGERPRINT_ATTEMPT"***. URL: http://www.whitehats.com/info/IDS5

[42] Whitehat's arachNIDS – The Intrusion Event Database. ***"SHELLCODE-X86-NOPS"***. URL: http://www.whitehats.com/info/IDS181

[43] Whitehat's arachNIDS – The Intrusion Event Database. ***"SHELLCODE-x86-STEALTH-NOP"***. URL: http://www.whitehats.com/info/IDS291

[44] Whitehat's arachNIDS – The Intrusion Event Database. ***"SHELLCODE-X86-SETUID0"***. URL: http://www.whitehats.com/info/IDS283

[45] Whitehat's arachNIDS – The Intrusion Event Database. ***"SHELLCODE-X86-SETGID0"***. URL: http://www.whitehats.com/info/IDS284

[46] David Oborn's GCIA Practical at URL: http://www.giac.org/practical/David_Oborn_GCIA.html

[47] McDonald, Terry [incidents.org list] LOGS GIAC GCIA Version 3.3 Practical Detect. URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/05/msg00090.html

[48] Whitehat's arachNIDS – The Intrusion Event Database. ***"NTPDX-BUFFER-OVERFLOW"***. URL: http://www.whitehats.com/info/IDS492

[49] SANS' description of the Adore Worm. URL: http://www.sans.org/y2k/adore.htm

[50] The Linux.Adore Worm. URL: http://www.dials.ru/english/inf/linux_adore.htm

[51] F-Secure's Virus description: Adore. URL: http://www.f-secure.com/v-descs/adore.shtml

[52] Reiter, Michael. Exploiting Loadable Kernel modules. URL: http://www.giac.org/practical/Michael_Reiter_GCIH.zip

[53] Stearns, William. Adore Worm detection and removal tool. URL: http://www.ists.dartmouth.edu/IRIA/knowledge_base/tools/adorefind.htm

[54] Symantec's description of the MyParty worm. URL: http://securityresponse.symantec.com/avcenter/venc/data/w32.myparty@mm.html

[55] Yackley, Matt. [incidents.org list] MyParty Trojan behaviour. URL: http://www.incidents.org/archives/intrusions/msg03040.html

[56] RFB port for VNC. URL: http://home.earthlink.net/~jknapka/vncpatch.html

[57] Sven Dietrich, Neil Long and David Dittrich – An analysis of the "shaft" distributed denial of service tool. URL: http://home.adelphi.edu/~spock/shaft_analysis.txt

[58] Whitehat's arachNIDS – The Intrusion Event Database. ***"DDOS-SHAFT-CLIENT-TO-HANDLER"***. URL: http://www.whitehats.com/info/IDS254

[59] CERT Incident Note IN-2000-05 – "mstream" distributed denial of service tool. URL: http://www.cert.org/incident_notes/IN-2000-05.html

[60] David Dittrich, George Weaver, Sven Dietrich and Neil Long – The "mstream" distributed denial of service tool. URL: http://staff.washington.edu/dittrich/misc/mstream.analysis.txt

[61] Ruiu, Dragos [neohapsis archives] Incomplete Packet Fragments Discarded? URL: http://archives.neohapsis.com/archives/snort/2001-02/0320.html

[62] The TFTP Protocol (Revision 2). URL: http://www.rfc-editor.org/rfc/rfc1350.txt

[63] IRC bot and backdoor: XDCC. URL: http://security.duke.edu/cleaning/xdcc.html

[64] Symantec's description of the SdBot backdoor Trojan. URL: http://securityresponse.symantec.com/avcenter/venc/data/backdoor.sdbot.html

[65] Brandon Newport's GCIA Practical. URL: http://www.giac.org/practical/Brandon_Newport_GCIA.zip