Intrusion Detection and Analysis Theory, Techniques, Tools and Detects

Reto Baumann

GIAC GCIA Practical (Version 3.3)
Submitted: May, 2003

Table of Contents

Conventions Used in this Paper

Normal text is written in 12-point Arial. A lot of command-line command are used, they look the following

```
$ command to issue on a shell, the '$' indicates the
command-line prompt
```

A lot of mysql commands are also used, the look similar to the command used on a shell

```
Mysql> Here comes a mysql command
```

Due to the heavy use of log entries, they have a somewhat smaller appearance

```
Logs are written in 9-point Courier-New
```

Output from all commands is enclosed in a box as if it would be a screenshot

```
Output from a command is surrounded by a box and therefore treated
like a screenshot. ASCII art should also appear correctly
        +---------------------+
        | Small Box in ASCII  |
        +---------------------+
```

Assignment #1: Describe the State of Intrusion Detection

Honeyd – A Low Involvement Honeypot

Introduction

Honeyd – What could that be? Well, honeyd is a small little program with a great effect – you can spend hours of watching and fine-tuning honeyd and the associated scripts and it is even fun. Honeyd is an application which enables the setup of multiple virtual honeypots on a single machine, each with different characteristics and services. But let's start at the beginning, let's first have a look at the honeypot technology before we are coming back with more details for honeyd.

What is a Honeypot

Honeypots aren't something that new – the basic idea of a honeypot is quite old and was used already for quite a long time. Although, the word "Honeypot" is a new one and the technology is getting more and more important. But let's first have a look at a possible definition of what a honeypot is

"A honeypot is a resource which pretends to be a real target. A honeypot is expected to be attacked or compromised. The main goals are the distraction of an attacker and the gain of information about an attack and the attacker." - R. Baumann, C. Plattner

A honeypot therefore is a system which is acting as a potential target for an attacker. The system itself though isn't of much value to the operator as no valuable information or important services are located on that machine – it's the opposite. All services running on a honeypot aren't used in the productive environment. The services aren't promoted and so there shouldn't be any productive traffic going for these systems. Due to this fact, all traffic heading and reaching a honeypot is of potential value and should be analyzed. A honeypot doesn't need to deal with false positives like an intrusion detection system as there are simply no false positives – all traffic is suspicious as there shouldn't be any traffic because nobody knows of the system, no productive services are running and the system is not involved in "normal" activities.

Two Honeypot Categories

Two categories of honeypots have evolved – research and productive honeypots. Research honeypots are used primarily for research activities like detecting new kind of attacks, retrieving new hacker tools or to get a better knowledge about the attackers, their background, activities and goals. Research honeypots are valuable for developing new IDS signatures, analyze new attack tools or detect new ways of hidden communications or distributed denial of service (DDoS) tools. Research honeypots normally have great logging capabilities to log a hacker's activity once the attacks started or he gained root access.

The other category, the productive honeypots, is mostly used to distract an attacker from the real target. A honeypot is used as a bait to bind his attacking attempts as long as possible to the unproductive honeypot in order to gain time and protect the productive environment in the meantime. A productive honeypot is primarily not interested in gaining new knowledge about the blackhat community – its main interest is the protection of the real servers. Productive honeypots sometimes are also used to gather enough evidence for a successful prosecution of a hacker – But this application is still controversial and the legal side of such procedures is also not clear.

Level of Involvement

Besides the two usage categories of honeypots we already seen, there are also three different technical implementations of honeypots. The essential factor to distinguish here is the "level of involvement". A honeypot is acting like a "normal" server to the attacker – he offers certain services on different ports and could have certain vulnerabilities. Depending on the usage of a honeypot, having some real services on that machine is not always desired or even needed. It could be enough to have a simple listener bound to a port which just writes all incoming packets to a file and never answers to the received request. For catching an infected Microsoft Internet Information Server this is enough, no real IIS is needed. On the other hand, to study a hacker's social network and ways of communicating it could be necessary to "offer a real shell" and allow the attacker to gain root privileges. Once a hacker is root on a system it could be very interesting to see what he's going to do and for what he does need his newly gained system. These different honeypots can be described with the level of involvement

- Low involvement: They are listening on a certain port for incoming connections. All packets are logged. No answer to the request is sent. Low involvement honeypots have no interaction with the attacker. No traffic is ever leaving the honeypot – It's a simple logging machine.
- Mid involvement: Mid involvement honeypots also listen on different ports. But in contradiction to low involvement they send information back to the attacker. A request is answered and the attacker has the possibility to issue commands. Normally, mid involvement honeypots don't use real daemons, instead scripts or small programs are used to imitate the behavior of a service. The provided functionality depends on the script – in most cases, the provided commands are very limited. The big advantage of using such scripts is their logging capabilities and the circumvention of possible vulnerabilities of real services.
- High involvement: High involvement honeypots are the most advanced honeypots. They use real daemons and provide the full set of functionality. An attacker can do whatever he could do to a productive system – no limitations in functionality, vulnerability or behavior. Unfortunately, logging all attempts with high details isn't always easy and the risk of a compromise is growing. Mostly, high involvement honeypots are used when a compromise of a system is desired.

The following table (source "Honeypots" by R. Baumann, C. Plattner) provides an overview of the different honeypot and their level of involvement

	Low Inv.	**Mid Inv.**	**High Inv.**
Degree of involvement	Low	Mid	High
Real operating system	-	-	x
Risk	Low	Mid	High
Information gathering	Connections	Requests	All
Compromised wished	-	-	x
Knowledge to run	Low	Low	High

Knowledge to develop	Low	High	High
Maintenance time	Low	Low	Very high

Even if the idea of honeypots is not that new, honeypots are still in its infancy. Especially the combination of honeypots with intrusion detection systems and firewalls could open some new possibilities in the fight against the blackhat community. It is possible that the combination of honeypots and intrusion detection could be used to dynamically update firewall rules and actively prevent attacks before they can be targeted at a productive system.

This short description of what a honeypot is, what kind of honeypots there are and how they are setup technically should provide enough background to go on with honeyd.

Honeyd – A Virtual Honeypot

Honeyd is a freely available framework for setting up virtual honeypots. With honeyd it is possible to setup honeypots with different personalities and services on one machine. Honeyd emulates the different operating system's IP stack and binds certain script to a desired port to emulate a specific service. Honeyd is able to fool network fingerprinting tools to think they are dealing with a real operating system ranging from a Windows NT to an AIX box. Even different router's IP stacks can be emulated. Honeyd relies on the nmap fingerprinting file which is used to characterize different kind of operating systems and their IP stacks. Before honeyd is inserting a packet into the IP stream, the personality of the packet is adjusted according to the desired operating system and the corresponding TCP/IP flags. With honeyd it is even possible to emulate complex network architectures and their characteristics. Virtual routing topologies can be defined including different brands of routers, the latency of a network connection as well as the packet loss. When using tools to map the network (like traceroute), the network traffic appears to follow the configured routers and network connections.
The setup of virtual machines is very easy. A configuration file is used to tell honeyd what kind of operating system is desired, how it does respond to closed ports and what kind of service is listening on which port. Honeyd is capable of binding a script to a network port. The script can be a standard shell script which simulates a certain service. Most scripts are built as state machines where a command triggers a certain response or advances to a new state with new possibilities. Scripts for the most popular well known services like SMTP, HTTP and telnet are available at several locations on the Internet.

Honeyd Installation and Configuration

The honeyd installation is straight forward and no problem at all. All you have to get first are three libraries

- libevent - an asynchronous event library
- libnet – a network library
- libpcap - a packet capture library

Installing all three libraries is very easy; just go the usual way of installing a UNIX application with

```
     $ ./configure; make; make install
```

Honeyd itself is also installed in the same way. After successfully installing all components it is a wise idea to store all honeyd scripts in a central place.

As mentioned, honeyd is configured via a simple text file where all virtual honeypots as well as the virtual network topology is specified. Each system is first created with a `create` command. The system then is further specified and configured with `add` and `set` commands. With the `set personality` command, a personality is assigned to a created system. It is further possible to choose the default action for the supported network protocols like `block`, `reset` or `open`. If the default value is set to be `open`, all ports for the desired protocol are in a listening state. The value `reset` defines all ports to be closed while `block` is used to drop all packets for the designated protocol.
Adding services, therefore binding scripts to a certain port, is done by using the `add` command. Instead of binding a script to a port it is also possible to forward the traffic to another IP by using the keyword `proxy`. Honeyd defines four variables which can be very handy: `$ipsrc` for the IP source, `$sport` for the source port, `$ipdst` for the targeted IP and finally `$dport` for the destination port. With these variables it is possible to pass parameters to the scripts or to forward traffic based on one of these values.
The created systems are then assigned to an IP with the `bind` command. The following example configuration file shows most of these commands as well as a basic example for defining a network topology.

```
route entry 192.168.1.1
route 192.168.1.1 link 192.168.1.0/24
route 192.168.1.1 add net 192.168.2.0/24 192.168.2.1 latency 45ms
loss 0.2
route 192.168.1.1 add net 192.168.3.0/24 192.168.3.1 latency 10ms
loss 0.1
route 192.168.2.1 link 192.168.1.0/24
route 192.168.3.1 link 192.168.2.0/24

create router
set router personality “Cisco 7206 running IOS 11.1(24)”
set router default tcp action  reset
add router tcp port 23 “scripts/router-telnet.pl”
bind 192.168.1.1 router

create linux
set linux personality "Linux 2.2.12 - 2.2.19"
add linux tcp port 23 "sh scripts/telnet.sh"
add linux tcp port 22 open
set linux uptime 112211
set linux default tcp action reset
set linux default udp action reset
bind 192.168.1.2 linux
```

The example configuration script generates a simple network topology as well as a Cisco router with telnet running on port 23 as well as a Linux operating system with telnet on port 23 and an open ssh port. The Linux system is set to have an uptime of 112211 seconds.
As it can be seen here, adding new systems, modifying existing ones and even construct virtual networks is straight forward and easily achieved.

Practical Results

To see honeyd in action, I decided to give it a try and configure three different systems with different services with honeyd. The following chapters will describe how honeyd was installed and configured and most importantly, what kind of attacks did hit honeyd during a two weeks testing period.

Installation and Configuration

Honeyd was installed as described in one of the earlier chapters. To be able to catch and categorize most attacks I also installed Snort in its latest version available at the time of writing (version 1.9.1) as well as MySQL 4.1 for logging all Snort events to the database for easier analysis. Describing the installation of all involved pieces of software would break the content here, but installing them shouldn't be a problem for a UNIX user/administrator.

One physical system was configured with honeyd to host three virtual machines. Each should listen on its own IP address. Honeyd is using libpcap to listen on a network interface and to capture the traffic. To have traffic for the configured IP's passing our network interface, we have to answer the corresponding ARP requests. For this purpose, arpd was installed. This little tool listens on an interface and answers ARP requests for some desired IP addresses. With the help of arpd it was ensured that the traffic for our virtual honeypots did get to our physical interface.

For honeyd, the following configuration file was used:

```
annotate "AIX 4.0 - 4.2" fragment old
create aix
set aix personality "AIX 4.0 - 4.2"
add aix tcp port 80 "sh scripts/web.sh"
add aix tcp port 22 "sh scripts/test.sh $ipsrc $dport"
set aix default tcp action reset
bind 10.0.0.2 aix

create linux
set linux personality "Linux 2.2.12 - 2.2.19"
add linux tcp port 23 "sh scripts/telnet.sh"
add linux tcp port 22 open
set linux uptime 112211
set linux default tcp action reset
set linux default udp action reset
bind 10.0.0.3 linux

create windows
set windows personality "Windows NT 4.0 Server SP5-SP6"
set windows default tcp action reset
set windows default udp action reset
add windows tcp port 80 "perl scripts/iisemulator-0.95/iisemul8.pl"
add windows tcp port 139 open
add windows tcp port 137 open
add windows udp port 137 open
add windows udp port 135 open
set windows uptime 42002
bind 10.0.0.4 windows
```

For Snort, the following configuration file was used

```
#--------------------------------------------------
#   http://www.snort.org     Snort 1.9.1 Ruleset
#     Contact: snort-sigs@lists.sourceforge.net
#  Ruleset for honeyd field project
#--------------------------------------------------
# NOTE:This ruleset only works for 1.9.1 and later
#--------------------------------------------------
#
###################################################
# Step #1: Set the network variables:
#
var HOME_NET 62.2.201.201/28
var EXTERNAL_NET !62.2.201.201/28
var DNS_SERVERS $HOME_NET
var SMTP_SERVERS $HOME_NET
var HTTP_SERVERS $HOME_NET
var SQL_SERVERS $HOME_NET
var TELNET_SERVERS $HOME_NET
var HTTP_PORTS 80
var SHELLCODE_PORTS !80

# Path to your rules files (this can be a relative path)
var RULE_PATH /etc/snort/rules

###################################################
# Step #2: Preprocessors
#
preprocessor frag2
preprocessor stream4: detect_scans, disable_evasion_alerts
preprocessor stream4_reassemble
preprocessor http_decode: 80 unicode iis_alt_unicode double_encode
iis_flip_slash full_whitespace
preprocessor rpc_decode: 111 32771
preprocessor bo: -nobrute
preprocessor telnet_decode
preprocessor portscan: $HOME_NET 4 3 portscan.log
preprocessor conversation: allowed_ip_protocols all, timeout 60,
max_conversations 32000

####################################################################
# Step #3: Configure output plugins
#
output log_tcpdump: tcpdump.log
output alert_full: snort-alerts.txt
output database: log, mysql, user=snort password=snort dbname=snort
host=localhost

####################################################################
# Step #4: Customize your rule set
#
# The standard rules where used… but removed here to save space
```

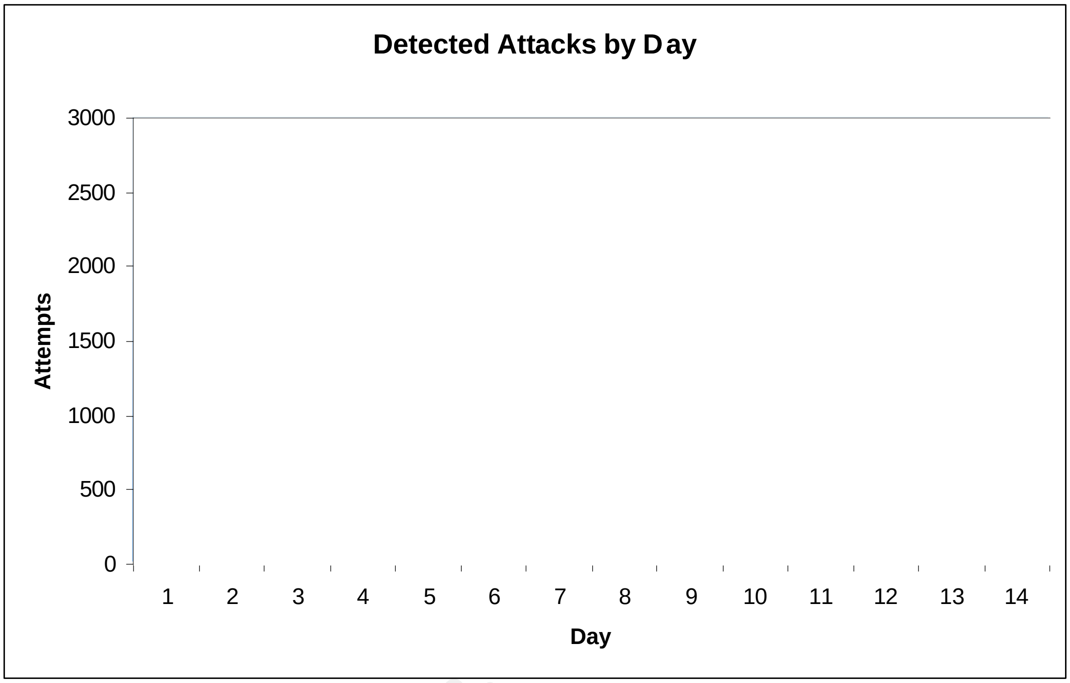
After starting Snort and arpd, the system was ready to host three new virtual honeypots. Honeyd was then started

```
$ honeyd -p nmap.prints -f honeyd.conf -x xprobe2.conf
-a nmap.assoc –l /usr/local/honeyd/logs 10.0.0.2-
10.0.0.4
```

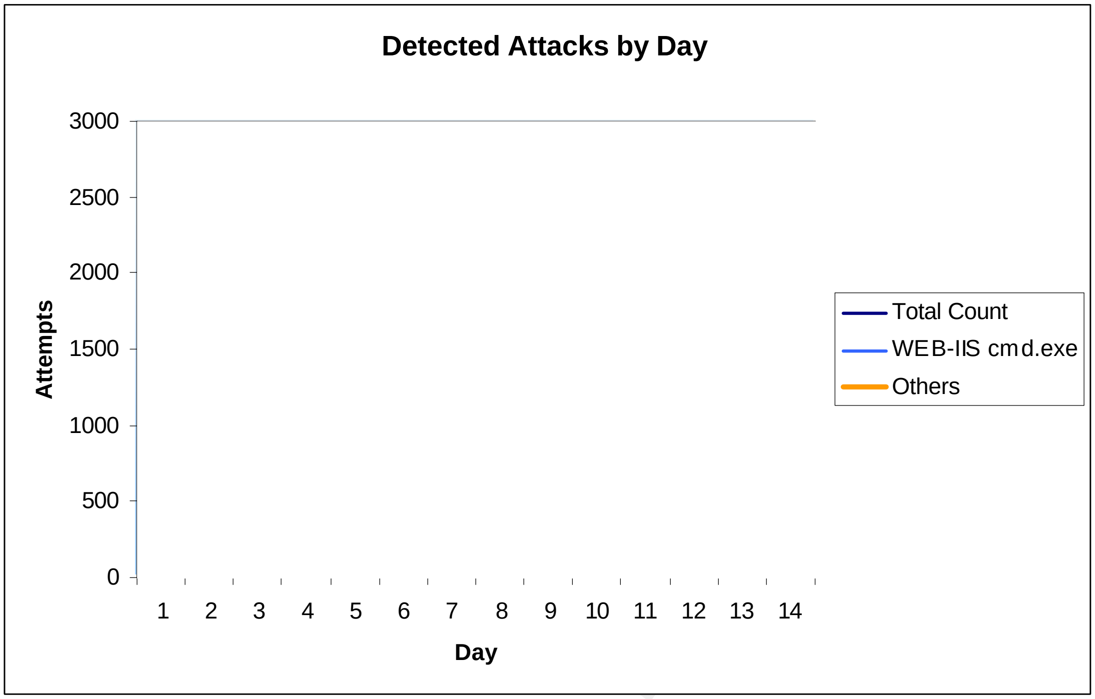
After checking the readiness of our honeypots, the system was left alone for 14 days to gather and log attacks destined for its victims.

Results

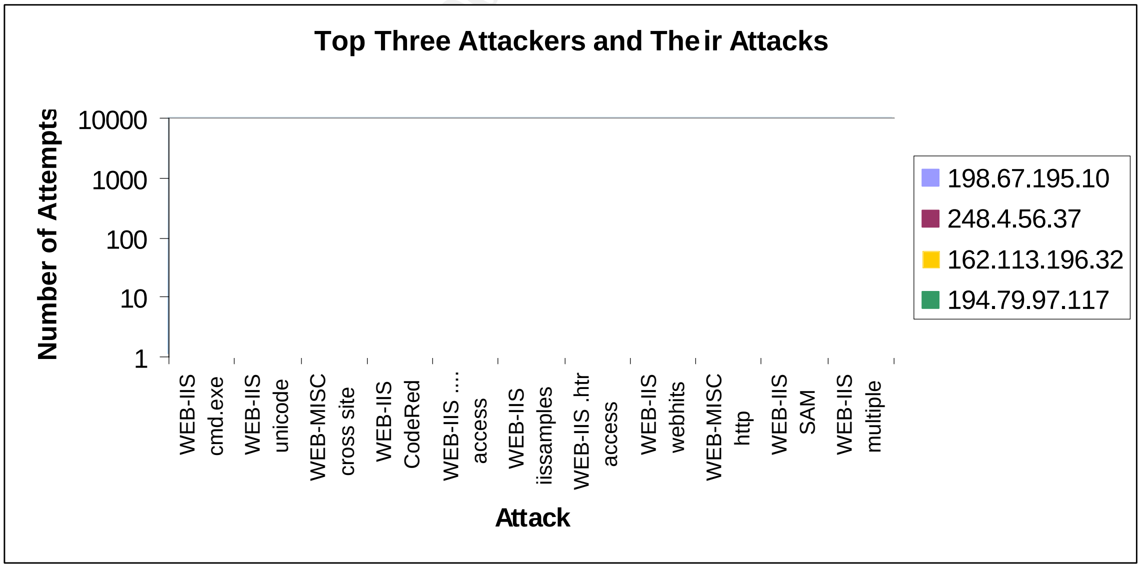
The honeyd configuration was running for exactly 14 days. All events were stored in a MySQL database for easy statistical analysis. In the two weeks, 11121 alerts were generated. It would be interesting to see if we have about the same amount of attacks each day.



As it can be easily seen, we have huge differences in the number of attacks detected each day. By having a look at the alerts itself we can see that the signature "WEB-IIS cmd.exe access" triggered most of the alerts (more than 50%). Doing another chart where we explicitly show the "WEB-IIS cmd.exe access" alerts leads to some new conclusions. As it can be seen, all other alerts don't lead to such strong peaks which we had on day 2, 5 and 8. We still have some kind of "top-scorer days" but they are not as significant as others.

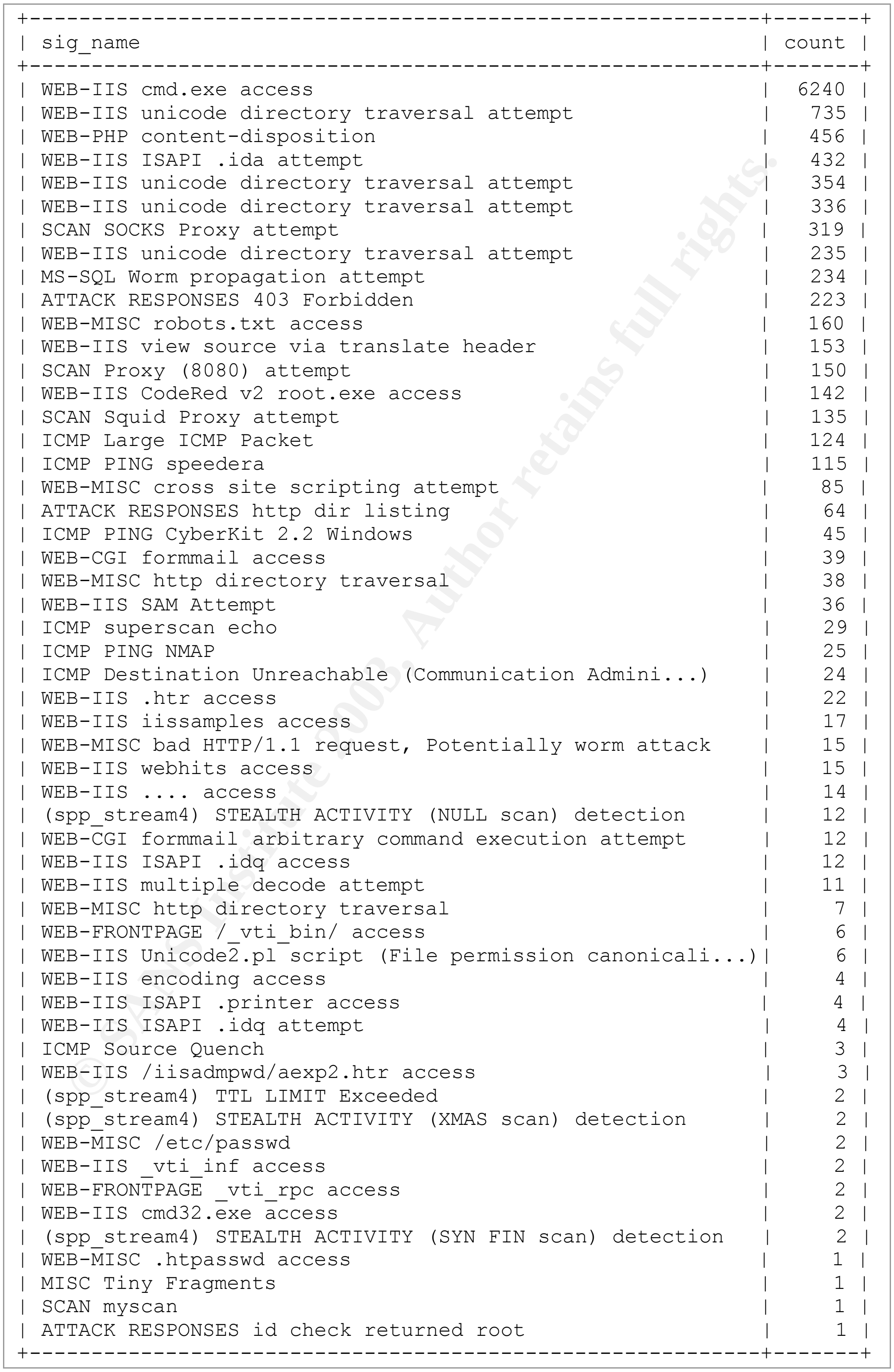


The next analysis which we were interested in was if we had top attackers and if so what they did. It's interesting to see that three attackers generated about 35% of all alerts. By examining these three attackers a little bit closer we can see that the mostly launched web attacks.



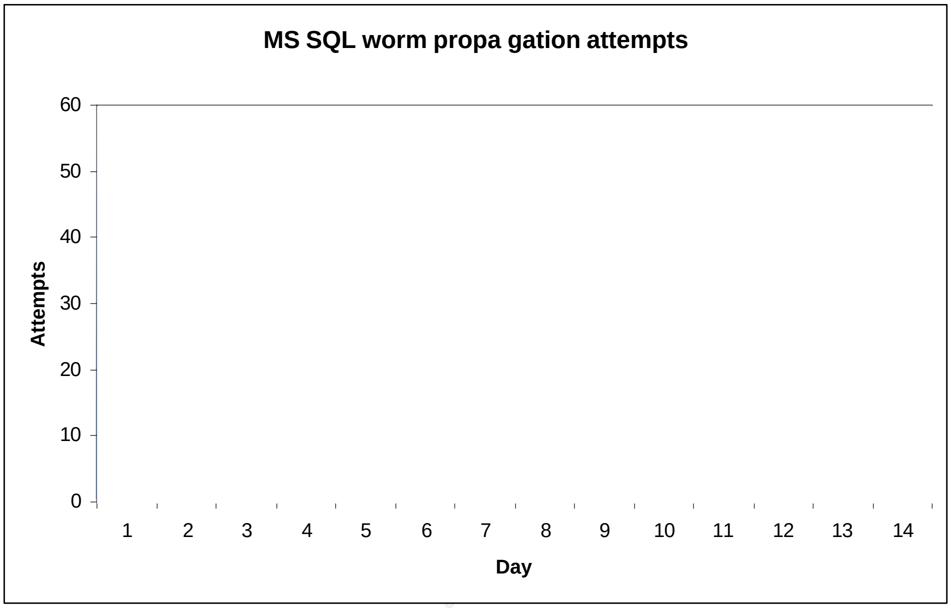
Querying the MySQL database about the alerts reveals the following table

```
mysql> select s.sig_name, count(*) as count from event
e, signature s where e.signature=s.sig_id group by
e.signature order by count desc;
```



```
+------------------------------------------------------------+-------+
| sig_name                                                   | count |
+------------------------------------------------------------+-------+
| WEB-IIS cmd.exe access                                     |  6240 |
| WEB-IIS unicode directory traversal attempt                |   735 |
| WEB-PHP content-disposition                                |   456 |
| WEB-IIS ISAPI .ida attempt                                 |   432 |
| WEB-IIS unicode directory traversal attempt                |   354 |
| WEB-IIS unicode directory traversal attempt                |   336 |
| SCAN SOCKS Proxy attempt                                   |   319 |
| WEB-IIS unicode directory traversal attempt                |   235 |
| MS-SQL Worm propagation attempt                            |   234 |
| ATTACK RESPONSES 403 Forbidden                             |   223 |
| WEB-MISC robots.txt access                                 |   160 |
| WEB-IIS view source via translate header                   |   153 |
| SCAN Proxy (8080) attempt                                  |   150 |
| WEB-IIS CodeRed v2 root.exe access                         |   142 |
| SCAN Squid Proxy attempt                                   |   135 |
| ICMP Large ICMP Packet                                     |   124 |
| ICMP PING speedera                                         |   115 |
| WEB-MISC cross site scripting attempt                      |    85 |
| ATTACK RESPONSES http dir listing                          |    64 |
| ICMP PING CyberKit 2.2 Windows                             |    45 |
| WEB-CGI formmail access                                    |    39 |
| WEB-MISC http directory traversal                          |    38 |
| WEB-IIS SAM Attempt                                        |    36 |
| ICMP superscan echo                                        |    29 |
| ICMP PING NMAP                                             |    25 |
| ICMP Destination Unreachable (Communication Admini...)     |    24 |
| WEB-IIS .htr access                                        |    22 |
| WEB-IIS iissamples access                                  |    17 |
| WEB-MISC bad HTTP/1.1 request, Potentially worm attack     |    15 |
| WEB-IIS webhits access                                     |    15 |
| WEB-IIS .... access                                        |    14 |
| (spp_stream4) STEALTH ACTIVITY (NULL scan) detection       |    12 |
| WEB-CGI formmail arbitrary command execution attempt       |    12 |
| WEB-IIS ISAPI .idq access                                  |    12 |
| WEB-IIS multiple decode attempt                            |    11 |
| WEB-MISC http directory traversal                          |     7 |
| WEB-FRONTPAGE /_vti_bin/ access                            |     6 |
| WEB-IIS Unicode2.pl script (File permission canonicali...)|     6 |
| WEB-IIS encoding access                                    |     4 |
| WEB-IIS ISAPI .printer access                              |     4 |
| WEB-IIS ISAPI .idq attempt                                 |     4 |
| ICMP Source Quench                                         |     3 |
| WEB-IIS /iisadmpwd/aexp2.htr access                        |     3 |
| (spp_stream4) TTL LIMIT Exceeded                           |     2 |
| (spp_stream4) STEALTH ACTIVITY (XMAS scan) detection       |     2 |
| WEB-MISC /etc/passwd                                       |     2 |
| WEB-IIS _vti_inf access                                    |     2 |
| WEB-FRONTPAGE _vti_rpc access                              |     2 |
| WEB-IIS cmd32.exe access                                   |     2 |
| (spp_stream4) STEALTH ACTIVITY (SYN FIN scan) detection    |     2 |
| WEB-MISC .htpasswd access                                  |     1 |
| MISC Tiny Fragments                                        |     1 |
| SCAN myscan                                                |     1 |
| ATTACK RESPONSES id check returned root                    |     1 |
+------------------------------------------------------------+-------+
54 rows in set (0.11 sec)
```

Looks like most of our top ten attacks are well known web attacks. Nothing that interesting, except the “MS-SQL worm propagation”. We encountered 234 attempts of an MS-SQL worm propagation attempt. This would result in more than 15 attacks each day. Let’s check if we have a uniform distribution.



As it can be seen, we have days with much more attack attempts and days which no attempt was detected at all. Let’s see if all the attacks were from different IP addresses or if we are dealing with a handful of attackers (signature number 27 is our MS SQL worm propagation attempt).

```
mysql> select i.ip_src, count(*) as count from event e,
iphdr i where e.cid=i.cid and e.signature=27 group by
i.ip_src order by count desc;
```

```
+------------+-------+
| ip_src     | count |
+------------+-------+
| 2340602250 |    20 |
| 1093107862 |    13 |
| 3395763701 |     6 |
| 1029824543 |     5 |
| 3395788908 |     5 |
| 1078811399 |     5 |
| 3501812517 |     4 |
| 3256754466 |     3 |
| 1087796482 |     3 |
| 1061685509 |     3 |
...
```

The query reveals that we have two top attackers. Let’s check with whom we are dealing here. Our top attacker 139.130.193.138 (2340602250) belongs to an Australian company named Telstra, who is some kind of ISP offering Internet connectivity to their customers. The other IP, 130.79.1.22, belong to a university in France. I informed the administrator of the network about a possible infection of one of his MS SQL servers. The third one is an ISP again, this time from China.

Another attack which draws attention is the “ATTACK RESPONSES id check returned root”. Further investigation reveals that this must have been a false positive as it was an incoming packet destined at port 25 (SMTP). It looks as if a mail did contain information which triggered this alert.

Conclusion

Honeyd is a nice little tool which can be perfectly used to setup a low to mid involvement honeypot. The possibility to generate different virtual honeypots on one machine with even different simulated operating systems enhances the usability of this tool even further. It’s great for simulating victims and collecting a lot of interesting information. Honeyd could be used as a early warning system in a productive environment to catch some attacks and trigger an alert. Honeyd could also be very useful in detecting infected machines in a network by just sitting there and playing victim. Finding infected web or MS SQL servers can be achieved. The mechanism of attaching a script to a certain port allows a very flexible setup with unlimited capabilities and opportunities for tuning. Honeyd is not as flexible as a high involvement honeypot – the information that can be collected by a high involvement honeypot is much higher. On the downside, the associated risk and the amount of time needed for a good implementation is much higher. Honeyd is simple and the associated risk is very low – A tool which is very handy and much fun to run it.

Software Download Locations

I will shortly present a list of download locations for the different tools. Unfortunately, I can’t guarantee that they are still working at a later time. In case the source can’t be found there, try google (http://www.google.com) to search for them.

Tool	Download Location
honeyd	http://www.citi.umich.edu/u/provos/honeyd/
arpd	http://www.citi.umich.edu/u/provos/honeyd/
honeyd scripts	http://www.citi.umich.edu/u/provos/honeyd/contrib.html
libevent	http://www.monkey.org/~provos/libevent/
libdnet	http://libdnet.sourceforge.net/
libpcap	http://www.tcpdump.org/

References

Baumann, Reto and Plattner, Christian. “Honeypots”. 2002
URL: http://security.rbaumann.net/download/diplomathesis.pdf (March 14 2003)

Multiple authors. “The Honeynet Project”. 2003
URL: http://project.honeynet.org/ (March 10 2003)

Neville, Alan. “IDS Logs in Forensics Investigations: An Analysis of a Compromised Honeypot”. 2003. URL: http://www.securityfocus.com/infocus/1676 (March 10 2003)

Provos, Niels. “Honeyd – A Virtual Honeypot Daemon”. 2003
URL: http://www.citi.umich.edu/u/provos/papers/honeyd-eabstract.pdf (March 11 2003)

Roesch and Green. “Snort User’s Manual Chapter 2: Writing Snort Rules”. 2003
URL: http://www.snort.org/docs/writing_rules/chap2.html (March 15 2003)

Spitzner, Lance. “Honeypots: Tracking Hackers Website”. 2003
URL: http://project.honeynet.org/ (March 11 2003)

Spitzner, Lance. “Open Source Honeypots, Part Two: Deploying Honeyd in the Wild”. 2003. URL: http://www.securityfocus.com/infocus/1675 (March 10 2003)

Assignment #2: Network Detects

Detect #1: DNS named version attempt

Trace Log

For this attack I will focus on the following Snort alert

```
[**] [1:1616:4] DNS named version attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/20-01:43:11.504488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33
type:0x800 len:0x48
210.195.43.5:3639 -> 46.5.36.163:53 UDP TTL:45 TOS:0x0
ID:5511 IpLen:20 DgmLen:58
Len: 38
[Xref => arachnids 278][Xref => nessus 10028]
```

Source of Trace

The network traffic from this detect if from the incidents.org site, namely http://www.incidents.org/logs/Raw/2002.5.20

This file is a "real world" binary log generated by snort and sanitized for use within the GCIA practical. What need to be known for this log file:

- binary mode logging
- Only the packet that violated a rule is in this log
- Unknown set of Snort rules
- IP address of the protected network is sanitized.
- Checksums are modified
- No ICMP, DNS, SMTP or web traffic.
- IP addresses belonging to non-local hosts are the real ones.

For more details have a look at http://www.incidents.org/logs/Raw/README

I'm going to start with gathering more information about the network architecture and topology where this detect happened. At this point I would like to thank Andre Cormier for his excellent network analysis of his detects which I will use as a kind of guideline.

Let's collect the source MAC addresses

```
$ tcpdump -neqr 2002.5.20 | cut -d ' ' -f2 | sort | uniq
```

This command gathers all involved MAC addresses which are acting as a source. Two MAC addresses were found, 0:0:c:4:b2:33 and 0:3:e3:d9:26:c0.

Let’s do the same with the destination MAC addresses

```
$ tcpdump -neqr 2002.5.20 | cut -d ' ' -f3 | sort | uniq
```

The same two MAC addresses were found, looks like these are the only active communication parties. According to the MAC addresses and the IEEE standard (http://standards.ieee.org/regauth/oui/oui.txt) this devices are manufactured by Cisco. So let's say we have the following setup:

Figure 1: Basic Network Setup

Let's collect the IP addresses which are coming from each Cisco device.

Destination IP addresses with source MAC 0:0:c:4:b2:33

```
$ tcpdump -neqr 2002.5.20 ether src 0:0:c:4:b2:33 | cut
-d ' ' -f 7 | cut -d \. -f 1-4 | sort -t \. -n | uniq
```

```
4.41.92.141
12.216.20.67
12.221.144.154
12.229.115.156
--- SNIP ---- we get about 103 IP addresses...
```

Destination IP addresses with source MAC 0:3:e3:d9:26:c0

```
$ tcpdump -neqr 2002.5.20 ether src 0:3:e3:d9:26:c0 |
cut -d ' ' -f 7 | cut -d \. -f 1-4 | sort -t \. -n |
uniq
```

```
46.5.0.80
46.5.1.166
--- SNIP --- about 89 addresses, all belonging to
             46.5.0.0/16
```

It looks like our topology which we did draw at the beginning is going to work. Let's add the corresponding MAC and IP addresses:

What can we check next? Let's have a look at the source and destination ports. Probably we can say something more about these Cisco devices or about some filtering of network traffic. How about the destination ports coming in from Cisco-Dev #1?

```
$ tcpdump -neqr 2002.5.20 ether src 0:3:e3:d9:26:c0 |
cut -d ' ' -f 7 | cut -d . -f5 | sort -n | uniq
```

```
21:
53:
80:
515:
1080:
3128:
--- SNIP --- and some more ports, lots of em in the 60'000 range
```

Is CISCO-Dev#1 doing some ingres filtering? Hard to say, as we can't know if this is only the traffic which was logged or really all incoming traffic. But it seems strange to me to have port 515 (printer spooler) open at the border. If CISCO-Dev#1 is a border router or firewall, the filtering needs improvement. But probably CISCO-Dev#1 belongs to the ISP and CISCO-Dev#2 is the first device which we are in control of.

Detect was Generated by

I used Snort version 1.9.1 build 231 (latest version at the time of writing) with the standard 1.9.1 ruleset. Setting the network information variables was achieved by supplying the corresponding variables by command line option

```
$ snort -c /etc/snort/snort.conf -d -e -l snort-logs/ -r
2002.5.20 -S HOME_NET=46.5.0.0/16 -S
EXTERNAL_NET=\!46.5.0.0/16
```

```
Log directory = snort-logs/
TCPDUMP file reading mode.
Reading network traffic from "2002.5.20" file.
snaplen = 1514

        --== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file /etc/snort/snort.conf
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++
--- SNIP ---
ule application order: ->activation->dynamic->alert->pass->log

        --== Initialization Complete ==--

-*> Snort! <*-
Version 1.9.1 (Build 231)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
Run time for packet processing was 1.94679 seconds
database: Closing connection to database "snort"


========================================================

Snort processed 4285 packets.
Breakdown by protocol:                Action Stats:

    TCP: 4251       (99.207%)         ALERTS: 231
    UDP: 34         (0.793%)          LOGGED: 231
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
========================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0          (0.000%)
    Control Packets:    0          (0.000%)
    Data Packets:       0          (0.000%)
========================================================
Fragmentation Stats:
Fragmented IP Packets: 1          (0.023%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
========================================================

TCP Stream Reassembly Stats:
   TCP Packets Used:      0          (0.000%)
   Reconstructed Packets: 0          (0.000%)
   Streams Reconstructed: 0
========================================================
```

By telling Snort to log into a MySQL database, getting some more information about the number of alerts and other statistics is quite easy. I extracted to information about the distribution of the alerts, therefore which alerts we had and how often.

```
mysql> select s.sig_name, count(*) as count from
signature s, event e where e.signature=s.sig_id group by
e.signature order by count desc;
```

```
+---------------------------+-------+
| sig_name                  | count |
+---------------------------+-------+
| SHELLCODE x86 inc ebx NOOP |    48 |
| BACKDOOR Q access         |    47 |
| SCAN SOCKS Proxy attempt   |    40 |
| DNS named version attempt  |     34 |
| SHELLCODE x86 NOOP         |     28 |
| SHELLCODE x86 NOOP         |     23 |
| SCAN nmap TCP              |      4 |
| SCAN Squid Proxy attempt   |      2 |
| SCAN Proxy (8080) attempt  |      2 |
| SCAN FIN                   |      2 |
| BAD TRAFFIC bad frag bits  |      1 |
+----------------------------+--------+
```

Let's focus on 'DNS named version attempt'. The alert looks the following way:

```
[**] [1:1616:4] DNS named version attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/20-01:43:11.504488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33
type:0x800 len:0x48
210.195.43.5:3639 -> 46.5.36.163:53 UDP TTL:45 TOS:0x0 ID:5511
IpLen:20 DgmLen:58 Len: 38
[Xref => arachnids 278][Xref => nessus 10028]
```

The signature which triggered this alert is located in the file dns.rules:

```
alert udp $EXTERNAL_NET any -> $DNS_SERVERS 53
(msg:"DNS named version attempt"; content:"|07|version"; nocase;
offset:12; content:"|04|bind"; nocase; offset: 12;
reference:nessus,10028; reference:arachnids,278;
classtype:attempted-recon; sid:1616; rev:4;)
```

The signature is built to trigger on incoming UDP packets which follows these conditions:

- Destination port 53 of the $DNS_SERVERS (default set to $HOME_NET)
- The payload matches "|07|version" where |07| marks 0x07 (match is not case sensitive) with an offset of 12 (i.e. start comparing at an offset of 12 bytes) and also matches "|04|bind" staring at offset 12 (also not case sensitive).

The alert tells us

- Packet was capture on the 20th June (which seems to be a little bit strange as the log file is called 2002.5.20 but let's ignore this)
- Ether type was IP (type:0x800)
- Capture length is 72 (0x48)
- UDP packet
- Source 210.195.43.5, port 3639
- Destination 46.5.36.163, port 53
- Time to live (TTL) is 45
- IP ID is 5511
- Datagram length is 58 (IP Header + UDP Header + data), which tells us that the payload has to be 58 - 20 (IP Header, no options) - 8 (UDP Header) = 30
- Two references are given, archNIDS 278 and Nessus 10028
- The alert is called 'DNS named version attempt'

Probability the Source Address was Spoofed

Let's check who attacked us and what systems were targeted. Querying mySQL for this is easy - the alert ‘DNS named version attempt’ has got the signature number 22 so let’s query for that one.

```
mysql> select e.timestamp, i.ip_src, i.ip_dst from event
e, iphdr I where e.sid=3 and e.cid=i.cid and
e.signature=22 order by ip_dst, timestamp;
```

```
+---------------------+------------+-----------+
| timestamp           | ip_src     | ip_dst    |
+---------------------+------------+-----------+
| 2002-06-20 06:48:30 | 3418711760 | 772080285 |
| 2002-06-20 07:23:24 | 3413782409 | 772081974 |
| 2002-06-20 13:31:27 | 3413782409 | 772082150 |
| 2002-06-20 09:15:50 | 3415993744 | 772083449 |
| 2002-06-20 04:35:43 | 3413782409 | 772088637 |
| 2002-06-20 02:43:11 | 3536005893 | 772088995 |
| 2002-06-20 13:20:47 | 3413782409 | 772089632 |
| 2002-06-20 05:29:31 | 3415993744 | 772090429 |
| 2002-06-20 09:42:15 | 3413782409 | 772092996 |
| 2002-06-20 03:59:34 | 3536005893 | 772093094 |
| 2002-06-20 07:55:44 | 3418711760 | 772096230 |
| 2002-06-20 06:14:16 | 3415993744 | 772099034 |
| 2002-06-20 06:01:49 | 3413782409 | 772099847 |
| 2002-06-20 04:04:34 | 3536005893 | 772100655 |
| 2002-06-20 10:21:35 | 3413782409 | 772102760 |
| 2002-06-20 06:13:04 | 3413782409 | 772105367 |
| 2002-06-20 12:41:14 | 3418711374 | 772106081 |
| 2002-06-20 04:41:58 | 3415993744 | 772114680 |
| 2002-06-20 10:50:58 | 3413782409 | 772115187 |
| 2002-06-20 13:19:52 | 3413782409 | 772117462 |
| 2002-06-20 06:26:42 | 3418711760 | 772117569 |
| 2002-06-20 08:51:40 | 3413782409 | 772118881 |
| 2002-06-20 06:57:54 | 3413782409 | 772124247 |
| 2002-06-20 14:13:59 | 3418711763 | 772124453 |
| 2002-06-20 07:25:16 | 3418711760 | 772124890 |
| 2002-06-20 05:33:36 | 3415993744 | 772128184 |
| 2002-06-20 04:52:51 | 3413782409 | 772131757 |
| 2002-06-21 01:37:39 | 3536005991 | 772132897 |
| 2002-06-20 06:26:47 | 3418711760 | 772134209 |
| 2002-06-20 08:28:45 | 3536006032 | 772135396 |
| 2002-06-20 10:14:25 | 3413782409 | 772137519 |
| 2002-06-20 04:48:13 | 3415993744 | 772137706 |
| 2002-06-20 05:39:06 | 3413782409 | 772138013 |
| 2002-06-20 09:04:16 | 3536006032 | 772143548 |
+---------------------+------------+-----------+
34 rows in set (0.00 sec)
```

Quite an amount of alerts. Let's change our query a little bit to get only the attackers and their number of attacks:

```
mysql> select i.ip_src, count(*) from event e, iphdr i
where e.sid=3 and e.cid=i.cid and e.signature=22 group
by ip_src;
```

```
+------------+----------+
| ip_src     | count(*) |
+------------+----------+
| 3413782409 |       15 |
| 3415993744 |        6 |
| 3418711374 |        1 |
| 3418711760 |        5 |
| 3418711763 |        1 |
| 3536005893 |        3 |
| 3536005991 |        1 |
| 3536006032 |        2 |
+------------+----------+
```

What can we say so far? It looks like this kind of alert is quite frequent. We have a top attacker and some others who “attack” from time to time. What seems most interesting is the fact, that no machine gets probed more than once (coincidence or planned?).

Spoofing UDP isn't difficult at all and can therefore be considered probable. But for this information gathering attempt, the attacker would like to get some response to see which version of bind we are running. In this specific attack, the IP source address is very likely not spoofed.

Description of the Attack

The attack intends to send a probe to a DNS server to get the version of the running name server as a result. This attack is for information gathering and often seen as a pre-attack probe, prior to an attempted overflow attack.

Attack Mechanism

"The BIND DNS server has a feature whereby its database contains a CHAOS/TXT record with the name 'VERSION.BIND'. If somebody queries this record, the version of the BIND software will be returned." (source ISS.net)
The attacker is primarily interested in the running version of bind to see if he's dealing with a vulnerable installation where he could get access by using a buffer overflow.

Correlations

The alert lists two references:

- ArachNIDS 278 (http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids278)
  "This event indicates that a remote user has attempted to determine the version of BIND running on a nameserver. This is often a pre-attack probe used to locate vulnerable servers running the named service."
  ArchNIDS lists some more references:
  CVE-1999-0009
  BugTraq 134 (Multiple Vendor BIND iquery buffer overflow Vulnerability)
  advICE 2000417
- Nessus 10028 is also listed as a reference.

The information gathering attempt here is widely known and discussed heavily in multiple forums on the Internet.

Evidence of Active Targeting

The attackers did only launch these 'named version probes' (no other attacks from them have been registered). We don't know if they probed our name servers or if they did just launch these queries to a random IP address. As far as we can tell from the alerts we have, no previous probes were launched at these probed systems. Most probably it's a general scan of a network for vulnerable DNS servers.

Looking at one attacker could probably reveal more:

```
+---------------------+------------+-------+------+-------+------+
| timestamp           | ip_src     | ip_id | ttl  | sport | len  |
+---------------------+------------+-------+------+-------+------+
| 2002-06-20 04:35:43 | 3413782409 | 40422 |   42 | 15181 |   38 |
| 2002-06-20 04:52:51 | 3413782409 | 60921 |   42 | 31834 |   38 |
| 2002-06-20 05:39:06 | 3413782409 | 54022 |   42 | 10628 |   38 |
| 2002-06-20 06:01:49 | 3413782409 | 23872 |   40 | 10911 |   38 |
| 2002-06-20 06:13:04 | 3413782409 | 45294 |   42 | 21863 |   38 |
| 2002-06-20 06:57:54 | 3413782409 | 44348 |   42 | 21430 |   38 |
| 2002-06-20 07:23:24 | 3413782409 | 14195 |   42 | 24233 |   38 |
| 2002-06-20 08:51:40 | 3413782409 | 56825 |   42 | 22222 |   38 |
| 2002-06-20 09:42:15 | 3413782409 | 56644 |   42 | 27518 |   38 |
| 2002-06-20 10:14:25 | 3413782409 | 30015 |   42 | 14610 |   38 |
| 2002-06-20 10:21:35 | 3413782409 | 38997 |   40 | 21556 |   38 |
| 2002-06-20 10:50:58 | 3413782409 |  9389 |   42 | 28273 |   38 |
| 2002-06-20 13:19:52 | 3413782409 |  1522 |   42 | 20087 |   38 |
| 2002-06-20 13:20:47 | 3413782409 |  2510 |   42 | 20969 |   38 |
| 2002-06-20 13:31:27 | 3413782409 | 13738 |   42 | 31341 |   38 |
+---------------------+------------+-------+------+-------+------+
```

By looking at these values it could very well be that the attacker is scanning a huge amount of IP addresses (not all belonging to us). At least I think there is an automated tool doing these scans as the IP IDs are incrementing quite fast (about 1000 IDs in less than a minute)

```
| 2002-06-20 13:19:52 | 3413782409 |  1522 |   42 | 20087 |   38 |
| 2002-06-20 13:20:47 | 3413782409 |  2510 |   42 | 20969 |   38 |
```

Severity

The severity is calculated according to this formula:

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

Each element is worth 1 to 5 points, and the arithmetic gives us a range of -8 to +8.

Criticality	The probes are meant to find vulnerable DNS servers	**5**
Lethality	This scan is not lethal, but if a vulnerable version should be found the lethality would be great - but that's another story. This is just an information gathering attempt.	**2**
System	We don't know if our servers are patched, but the vulnerability is quite old.	**3**
Network	Network countermeasures doesn't seem to be in place to prevent this information gathering attempt.	**1**

Severity would therefore be (5+2) - (3+1) = 3, which is not that low. If we would know more about our DNS server (and probably would know that we're running the latest

BIND version with a dummy version information in-place) we would get a much lower value. Keeping an eye on this attacker and this server would definitively be a good idea.

Defensive Recommendation

Check all DNS servers and install the latest patches to ensure a high security level. Configure bind not to give out it's real version – a dummy value should be return instead. By inserting an additional value into the /etc/named.conf should do the trick.

```
options {
        # The directory statement defines the name server´s
        # working directory
        directory "/var/named";

        # Let’s return a fake value for our version
       version "unknown";
}
```

Multiple Choice Test Question

When does the following signature trigger an alert? Assume that we are using default values for $DNS_SERVERS.

```
alert udp $EXTERNAL_NET any -> $DNS_SERVERS 53
(msg:"DNS named version attempt"; content:"|07|version"; nocase;
offset:12; content:"|04|bind"; nocase; offset: 12;
reference:nessus,10028; reference:arachnids,278;
classtype:attempted-recon; sid:1616; rev:4;)
```

a) As soon as we have incoming UDP packet targeted at one of our DNS servers
b) As soon as we have the payload '|04|bind' or '|07|version' with an offset of 12 bytes and the UDP packet is targeted at port 53
c) As soon as we have an incoming UDP packet heading for port 53 with a payload containing '|07|VERSION' and '|04|bind' with an offset of 12.
d) This signature never triggers an alert as offset 12 is to short to hold '|07|version' and '|04|bind'

The correct answer is (c) as both content strings have to be present to trigger the alert. Answer (a) is not correct as this is not the only requirement and if the variable $DNS_SERVERS is set to it’s default values, the signature is not specific to DNS servers but to all systems defined as $HOME_NET. Answer (b) looks quite fine, except

Posting to incidents.org

As requested, I posted my detect on the 03/11/2003 to the intrusions@incidents.org mailing list and was looking forward to get some feedback. The original posting can be found at http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00141.html.
What’s most interesting about the posting is the interest in a possibility that somebody or something is coordinating these DNS probes, as no system got probed twice. On 3/5/03 Beth Binde already mentioned this possibility in one of her detects (see http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00077.html). Bryce Alexander asked, if there is a possibility to prove or disprove the existence of a theoretical or coordinated attack?

With only having the limited set of network traffic it seems quite impossible to get into this theory much further. If we are really dealing with a coordinated attack, all the machines which are sending probes have to be controlled by a master, have to do some kind of negotiating between them to coordinate their probes or follow certain patterns of IP addresses to scan. We can't say much about some possible communication between all the scanners as this traffic isn't passing our sensors.
It doesn't look like the attackers using a certain pattern for scanning their IP addresses. Sometimes they increase the targeted IP, sometimes they decrease (seems quite random to me).

What about some other statistics? Let's have a detailed look at one attacker

```
+---------------------+------------+-------+------+-------+------+
| timestamp           | ip_src     | ip_id | ttl  | sport | len  |
+---------------------+------------+-------+------+-------+------+
| 2002-06-20 04:35:43 | 3413782409 | 40422 |   42 | 15181 |   38 |
| 2002-06-20 04:52:51 | 3413782409 | 60921 |   42 | 31834 |   38 |
| 2002-06-20 05:39:06 | 3413782409 | 54022 |   42 | 10628 |   38 |
| 2002-06-20 06:01:49 | 3413782409 | 23872 |   40 | 10911 |   38 |
| 2002-06-20 06:13:04 | 3413782409 | 45294 |   42 | 21863 |   38 |
| 2002-06-20 06:57:54 | 3413782409 | 44348 |   42 | 21430 |   38 |
| 2002-06-20 07:23:24 | 3413782409 | 14195 |   42 | 24233 |   38 |
| 2002-06-20 08:51:40 | 3413782409 | 56825 |   42 | 22222 |   38 |
| 2002-06-20 09:42:15 | 3413782409 | 56644 |   42 | 27518 |   38 |
| 2002-06-20 10:14:25 | 3413782409 | 30015 |   42 | 14610 |   38 |
| 2002-06-20 10:21:35 | 3413782409 | 38997 |   40 | 21556 |   38 |
| 2002-06-20 10:50:58 | 3413782409 |  9389 |   42 | 28273 |   38 |
| 2002-06-20 13:19:52 | 3413782409 |  1522 |   42 | 20087 |   38 |
| 2002-06-20 13:20:47 | 3413782409 |  2510 |   42 | 20969 |   38 |
| 2002-06-20 13:31:27 | 3413782409 | 13738 |   42 | 31341 |   38 |
+---------------------+------------+-------+------+-------+------+
```

This attacker is going very slow – we only get about 2-3 probes an hour. The probes also don't follow a certain pattern (like always at certain times of an hour. Even by looking at the IP IDs it seems that there is no certain pattern. Let's check if we get a pattern about the number of ID's in a certain amount of time.

Minutes	Amount of used IP IDs	IDs/min
20	20’000	1000
50	60’000	1200
20	35’000	1750
12	22’000	1800
45	60’000	1350
30	35’000	1200
90	105’000	1200
50	65’000	1300
30	40’000	1300

If the attacker is always incrementing his ID by one, it seems like he has quite some constant load. Would it be possible that this machine is constantly scanning other machines? I think this could be... If we check the other attacker, we see that these hold similar IDs per minute ratio (also around 1000-1300 IDs/min). Is this just a value which is

found every day? Or are all those machines scanners using the same tool for probing networks?
Proving if we are dealing with some coordinated attacks or not without having more information seems not possible, although it would be very interesting to dig in deeper and get some more information.

References

Binde, Beth. “GIAC GCIA Version 3.3 Practical Detect #2”. March 5, 2003
URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00077.html (March 11, 2003)

IEEE Organisation. “OUI’s”. 2003
URL: http://standards.ieee.org/regauth/oui/oui.txt (March 9 2003)

Roesch and Green. “Snort User’s Manual Chapter 2: Writing Snort Rules”. 2003
URL: http://www.snort.org/docs/writing_rules/chap2.html (March 11 2003)

Stevens, W. Richard. “TCP/IP Ilustrated, Volume I – The Protocols”. Addison Wesley, 1994

Detect #2: BAD TRAFFIC data in TCP SYN packet

Trace Log

For this detect, I will focus on the following alert:

```
[**] [1:526:4] BAD TRAFFIC data in TCP SYN packet [**]
[Classification: Misc activity] [Priority: 3]
08/27-19:14:34.524488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33
type:0x800 len:0x4E
216.33.87.10:2200 -> 138.97.18.88:53 TCP TTL:242 TOS:0x0
ID:34525 IpLen:20 DgmLen:64
******S* Seq: 0x153A6A5B  Ack: 0x0  Win: 0x800  TcpLen: 20
[Xref => url www.cert.org/incident_notes/IN-99-07.html]
```

Source of Trace

The network traffic for this detect was obtained from the incidents.org website, namely http://www.incidents.org/logs/Raw/2002.7.27 .

This file is a "real world" binary log generated by snort and sanitized for use within the GCIA practical. What need to be known for this log file:

- Binary mode logging
- Unknown set of Snort rules
- Only the packet that violated a rule is in this log
- IP address of the protected network is sanitized.
- Checksums are modified
- No ICMP, DNS, SMTP or web traffic.
- IP addresses belonging to non-local hosts are the real ones

For more details have a look at http://www.incidents.org/logs/Raw/README

I'm going to start with gathering more information about the network architecture and topology where these detect happened. The same approach is used as in detect #1, so I'm skipping some steps here. For a complete listing of the needed steps have a look at detect #1 and adjust the commands accordingly. After finishing the necessary steps, we can draw the following network diagram and insert the corresponding MAC and IP addresses. We have somewhat different network architecture than in detect #1 and detect #2. A third MAC address seems to participate as destination. According to the IEEE standard, devices with this new MAC address are manufactured by Western Digital.

What's about this Western Digital device?
We only have one packet going for that MAC address... We never see something coming back (which triggered an alert). What kind of packet is going there?

```
tcpdump -neqr 2002.7.27 ether dst 0:0:c0:6b:e9:c6
13:37:31.424488 0:3:e3:d9:26:c0 0:0:c0:6b:e9:c6 60:
  205.171.49.23.64890 > 138.97.18.219.1080: tcp 0
13:37:36.984488 0:3:e3:d9:26:c0 0:0:c0:6b:e9:c6 60:
  205.171.49.23.61333 > 138.97.18.219.3128: tcp 0
```

Both packets are coming from 205.171.49.23 (and therefore from CISCO-Dev#1). What alerts are associated with these two packets?

```
[**] SCAN Squid Proxy attempt [**]
08/27-13:37:36.984488 0:3:E3:D9:26:C0 -> 0:0:C0:6B:E9:C6
type:0x800 len:0x3C
205.171.49.23:61333 -> 138.97.18.219:3128 TCP TTL:36 TOS:0x0
ID:48497 IpLen:20 DgmLen:40
******S* Seq: 0x3CB957F9  Ack: 0x0  Win: 0xC00  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] SCAN SOCKS Proxy attempt [**]
08/27-13:37:31.424488 0:3:E3:D9:26:C0 -> 0:0:C0:6B:E9:C6
type:0x800 len:0x3C
205.171.49.23:64890 -> 138.97.18.219:1080 TCP TTL:36 TOS:0x0
ID:57402 IpLen:20 DgmLen:40
******S* Seq: 0x3CB957F9  Ack: 0x0  Win: 0xC00  TcpLen: 20

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

Looks like normal Proxy scans. Could be that this is a directly connected machine with a Western Digital network card or probably some other network device manufactured by Western Digital.

Let's have a look at the source and destination ports. Probably we can say something more about these Cisco devices or about some filtering of network traffic. Looking for the incoming ports from Cisco-Dev#1 reveals some ports.

```
$ tcpdump -neqr 2002.7.27 ether src 0:3:e3:d9:26:c0 |
cut -d ' ' -f 7 | cut -d . -f5 | sort -n | uniq
```

```
0:
53:
```

```
80:
103:
137:
515:
1080:
2052:
2564:
3128:
6346:
61235:
--- SNIP --- some more ports in the 60K range
64928:
64950:
```

The empty line (no associated ports) shows that there are some fragmented packets (which Snort does list in its output further down).

It's difficult to make an assumption about ingress filtering with only this amount of data. We can't say for sure, if these are the only ports which are allowed in or if these are only the ports we actually had traffic triggering an alert (and therefore have the packets in the RAW log file). Never the less, it seems that we have a lot of open ports on Cisco-Dev#1. The higher ports seem to be returning traffic to a host in our network. But the lower ports, i.e. 0, 53, 80, 103, 137, 515,1080 are some well known services. It can't be considered best practices to allow incoming traffic to all these services, especially to the 137 ports (NetBIOS) and port 515 (printer spooler). This could denote that this is a badly configured firewall or router or that this is the ISP's router which doesn't filter anything. In that case, Cisco-Dev#2 would probably be our border router or firewall. But what would be the Western Digital device part in this play? As we can see, it doesn't matter for our particular detect, so let's go on.

Detect was Generated by

I used Snort version 1.9.1 build 231 (latest version at the time of writing) with the standard 1.9.1 ruleset. Setting the network information variables was achieved by supplying the corresponding variables by command line option. It is always a good idea to set the variables to minimize false positives.

```
$ snort -c /etc/snort/snort.conf -d -e -l snort-logs/ -r
2002.7.27 -S HOME_NET=138.97.0.0/16 -S
EXTERNAL_NET=\!138.97.0.0/16
```

```
Log directory = snort-logs/
TCPDUMP file reading mode.
Reading network traffic from "2002.7.27" file.
snaplen = 1514

        --== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file /etc/snort/snort.conf

+++++++++++++++++++++++++++++++++++++++++++++++++++
Initializing rule chains...
--- SNIP ---
Rule application order: ->activation->dynamic->alert->pass->log
```

```
        --== Initialization Complete ==--

-*> Snort! <*-
Version 1.9.1 (Build 231)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
Run time for packet processing was 1.429801 seconds


===============================================================================

Snort processed 8912 packets.
Breakdown by protocol:                Action Stats:

    TCP: 8911       (99.989%)         ALERTS: 32
    UDP: 0          (0.000%)          LOGGED: 32
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
===============================================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0          (0.000%)
    Control Packets:    0          (0.000%)
    Data Packets:       0          (0.000%)
===============================================================================
Fragmentation Stats:
Fragmented IP Packets: 2         (0.022%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
===============================================================================

TCP Stream Reassembly Stats:
   TCP Packets Used:      0          (0.000%)
   Reconstructed Packets: 0          (0.000%)
   Streams Reconstructed: 0
===============================================================================
```

By telling Snort to log into a MySQL database, getting some more information about the number of alerts and other statistics is quite easy. I extracted to information about the distribution of the alerts, therefore which alerts we had and how often.

```
mysql> select s.sig_name, count(*) from signature s,
event e where e.signature=s.sig_id group by e.signature;
```

```
+-----------------------------------+----------+
| sig_name                          | count(*) |
+-----------------------------------+----------+
| BAD TRAFFIC tcp port 0 traffic    |        1 |
| SCAN nmap TCP                     |       16 |
| SCAN Squid Proxy attempt          |        1 |
| BAD TRAFFIC bad frag bits         |        1 |
| SCAN SOCKS Proxy attempt          |        5 |
| SCAN FIN                          |        1 |
| BAD TRAFFIC ip reserved bit set   |        1 |
| BAD TRAFFIC data in TCP SYN packet |       6 |
| BACKDOOR Q access                  |       30 |
| SHELLCODE x86 NOOP                 |        4 |
| SHELLCODE x86 NOOP                 |        6 |
| SHELLCODE x86 inc ebx NOOP         |       38 |
| Virus - Possible pif Worm          |        1 |
| Virus - Possible scr Worm          |        1 |
| SHELLCODE x86 stealth NOOP         |       15 |
+------------------------------------+----------+
15 rows in set (0.00 sec)
```

Let's have a look at the 'BAD TRAFFIC data in TCP SYN packet':

```
[**] [1:526:4] BAD TRAFFIC data in TCP SYN packet [**]
[Classification: Misc activity] [Priority: 3]
08/27-19:14:34.524488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33
type:0x800 len:0x4E
216.33.87.10:2200 -> 138.97.18.88:53 TCP TTL:242 TOS:0x0
ID:34525 IpLen:20 DgmLen:64
******S* Seq: 0x153A6A5B  Ack: 0x0  Win: 0x800  TcpLen: 20
[Xref => url www.cert.org/incident_notes/IN-99-07.html]
```

The signature which triggered this alert is located in the bad-traffic.rules file:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:"BAD TRAFFIC data in TCP SYN packet"; flags:S; dsize:>6;
reference:url,www.cert.org/incident_notes/IN-99-07.html;
sid:526; classtype:misc-activity; rev:4;)
```

The signature is constructed to alert as soon as any incoming TCP traffic (from any port to any port) has the SYN flag set and a payload size bigger than 6 bytes. The signature also lists a reference at www.cert.org which is a note about "Distributed Denial of Service Tools".

Let's dig deeper and have a detailed look at the alert and its information:

- Packet was capture on the 27th August (which seems to be a little bit strange as the log file is called 2002.7.27 but let's ignore this)
- Ether type was IP (type:0x800)
- Ether length is 78 (0x4e)
- TCP packet
- Source 216.33.87.10, port 2200
- Destination 138.97.18.88, port 53
- Time to live (TTL) is 242 (so the source is probably a Solaris machine)
- IP ID is 34525
- IP header length is 20
- TCP header length is 20
- Datagram length is 64 (IP Header + TCP Header + data), which tells us that the payload has to be 64 - 20 (IP Header, no options) - 20 (TCP Header) = 14
- The alert is called 'BAD TRAFFIC data in TCP SYN packet'

We have 78 alerts which triggered for this specific destination host. By querying MySQL we can retrieve a history of the alerts associated with the targeted host 138.97.18.88.

```
mysql> select s.sig_name, i.ip_src as src, t.tcp_sport
as sport, t.tcp_dport as dport from signature s, event
e, iphdr i, tcphdr t where i.ip_dst=2321617496 and
t.sid=3 and t.cid=e.cid and e.sid=3 and i.sid=3 and
e.cid=i.cid and e.signature=s.sig_id;
```

```
+------------------------------------+------------+-------+-------+
| sig_name                           | src        | sport | dport |
+------------------------------------+------------+-------+-------+
| SHELLCODE x86 NOOP                 | 1071192696 |    80 | 61852 |
| SHELLCODE x86 NOOP                 | 1071119175 |    80 | 61501 |
| SHELLCODE x86 inc ebx NOOP         |  208532768 |    80 | 62564 |
| SHELLCODE x86 inc ebx NOOP         | 1064246973 |    80 | 64881 |
| SHELLCODE x86 inc ebx NOOP         |  208532768 |    80 | 62667 |
| SHELLCODE x86 inc ebx NOOP         | 3499837013 |    80 | 64927 |
| SHELLCODE x86 inc ebx NOOP         | 3499837013 |    80 | 64928 |
| SCAN nmap TCP                      | 1094722172 |    80 |  2052 |
| SCAN nmap TCP                      | 1098526588 |    80 |  2052 |
| SHELLCODE x86 inc ebx NOOP         | 3492644228 |    80 | 62114 |
| SHELLCODE x86 inc ebx NOOP         | 3492644228 |    80 | 62119 |
| SHELLCODE x86 inc ebx NOOP         | 3492644228 |    80 | 62120 |
| SHELLCODE x86 inc ebx NOOP         | 3492644228 |    80 | 62121 |
| SCAN nmap TCP                      | 1070797284 |    80 |    53 |
| SCAN nmap TCP                      | 1070797284 |    53 |    53 |
| SCAN nmap TCP                      | 1083721284 |    80 |    53 |
| SHELLCODE x86 inc ebx NOOP         |  209192206 |    80 | 64747 |
| SHELLCODE x86 NOOP                 |  411084373 |  5729 | 63308 |
| SHELLCODE x86 inc ebx NOOP         |  209200398 |    80 | 61241 |
| SHELLCODE x86 inc ebx NOOP         | 3629358869 |    80 | 61582 |
| SHELLCODE x86 inc ebx NOOP         | 3499837013 |    80 | 61264 |
| SHELLCODE x86 inc ebx NOOP         | 3499837013 |    80 | 61264 |
| SHELLCODE x86 inc ebx NOOP         | 3286560913 |    80 | 64657 |
| SHELLCODE x86 inc ebx NOOP         | 3451704953 |    80 | 64859 |
| SHELLCODE x86 inc ebx NOOP         | 3451704953 |    80 | 64950 |
| Virus - Possible pif Worm          | 3515571458 |   110 | 61413 |
| Virus - Possible scr Worm          | 3515571458 |   110 | 61413 |
| SHELLCODE x86 NOOP                 | 3343908582 |    80 | 61576 |
| SHELLCODE x86 NOOP                 | 3343908582 |    80 | 61576 |
| SHELLCODE x86 stealth NOOP         | 1066396232 |    80 | 62416 |
| SHELLCODE x86 inc ebx NOOP         | 1110452630 |    80 | 63861 |
| SHELLCODE x86 NOOP                 | 2155506697 |    80 | 62071 |
| SHELLCODE x86 NOOP                 | 3487885576 |    80 | 63980 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 64637 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 64637 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 64637 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 64637 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 64637 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 64637 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 64637 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 63067 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 63067 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 63067 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 63067 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 63067 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 63067 |
| SHELLCODE x86 stealth NOOP         | 3414095556 |    80 | 63067 |
| SHELLCODE x86 NOOP                 | 3487930411 |    80 | 62416 |
| SHELLCODE x86 inc ebx NOOP         | 3333382177 |    80 | 62949 |
| SHELLCODE x86 inc ebx NOOP         | 2704986086 |    80 | 63400 |
| BAD TRAFFIC data in TCP SYN packet | 3626063626 |  2200 |    53 |
| BAD TRAFFIC data in TCP SYN packet | 3626063626 |  2202 |    53 |
| BAD TRAFFIC data in TCP SYN packet | 3626063626 |  2201 |    53 |
| SHELLCODE x86 inc ebx NOOP         | 3451704889 |    80 | 63240 |
| SHELLCODE x86 inc ebx NOOP         | 3631134411 |    80 | 62833 |
| SHELLCODE x86 inc ebx NOOP         | 3631134411 |    80 | 62844 |
| SHELLCODE x86 inc ebx NOOP         | 2204623863 |    80 | 61235 |
| SHELLCODE x86 inc ebx NOOP         | 2204623863 |    80 | 61382 |
| SHELLCODE x86 inc ebx NOOP         | 2204623863 |    80 | 61386 |
| SHELLCODE x86 NOOP                 | 3499620268 |  1671 | 61964 |
| SHELLCODE x86 NOOP                 | 3499620268 |  1671 | 61964 |
| SHELLCODE x86 inc ebx NOOP         |  209200398 |    80 | 61379 |
| SCAN nmap TCP                      | 1126454277 |    80 |  2564 |
| SCAN nmap TCP                      | 3331737861 |    80 |  2564 |
| SHELLCODE x86 inc ebx NOOP         | 3624910818 |    80 | 63377 |
| SHELLCODE x86 inc ebx NOOP         | 3624910818 |    80 | 63380 |
| SHELLCODE x86 inc ebx NOOP         | 3624910818 |    80 | 63381 |
| SHELLCODE x86 inc ebx NOOP         | 3624910818 |    80 | 63457 |
| SHELLCODE x86 inc ebx NOOP         | 3624910818 |    80 | 63460 |
| SHELLCODE x86 inc ebx NOOP         | 3624910818 |    80 | 63461 |
| SHELLCODE x86 inc ebx NOOP         | 3624910818 |    80 | 63476 |
| SHELLCODE x86 inc ebx NOOP         | 3624910818 |    80 | 63479 |
| SHELLCODE x86 inc ebx NOOP         | 3624910818 |    80 | 63480 |
| SHELLCODE x86 inc ebx NOOP         | 3435828240 |    80 | 63742 |
| BAD TRAFFIC data in TCP SYN packet | 3510836489 |  2100 |    53 |
| BAD TRAFFIC data in TCP SYN packet | 3510836489 |  2101 |    53 |
| BAD TRAFFIC data in TCP SYN packet | 3510836489 |  2102 |    53 |
| SHELLCODE x86 inc ebx NOOP         | 2833956637 |    80 | 63144 |
+------------------------------------+------------+-------+-------+
78 rows in set (0.00 sec)
```

Quite interesting. There seem to be a lot of alerts associated with this host. Checking the logs shows that we have no alerts associated with traffic leaving 138.97.18.88.

Taking a closer look at some of these 'SHELLCODE x86' reveals that they are false positives as this seems to be valid HTTP traffic (content type image/jpeg). It is known for these signatures to generate a lot of false positives as they are looking for a certain content in the packets which is quite common in file transfers. At least we can say, that somebody is using our machine 138.97.18.88 to browse some web pages (or at least receive some images). The fact that we also have some virus alerts could further harden the assumption that this is a workstation of some sort.

Let's query MySQL to return the data payload associated with these 'BAD TRAFFIC data in TCP SYN packet':

```
+----------------+--------------------------------------------------+
| timestamp      | data_payload                                     |
+----------------+--------------------------------------------------+
| 08-27 20:14:34 | 000000000000000000000000000000000000000000000000 |
| 08-27 20:14:34 | 000000000000000000000000000000000000000000000000 |
| 08-27 20:14:34 | 000000000000000000000000000000000000000000000000 |
| 08-28 01:39:23 | 000000000000000000000000000000000000000000000000 |
| 08-28 01:39:23 | 000000000000000000000000000000000000000000000000 |
| 08-28 01:39:23 | 000000000000000000000000000000000000000000000000 |
+----------------+--------------------------------------------------+
```

This looks rather interesting (also a bit disappointing). The payload of all these packets doesn't seem to make much sense. All packets' payload is only consisted of zeroes. SYN packets with payload are often used by DDoS tools to communicate with their master and to get new instructions. It seems as if these wouldn't make much sense here as we're not having some meaningful payload.

It could be important to realize, that payload which is sent before the three-way handshake is completed is appended to the payload once the session is established. Data sent in this way is not necessarily discarded. What possibilities or usages for this payload could be possible?

- Some kind of error
- Some kind of padding
- Hidden command for a tool
- Reconnaissance
- Some kind of 'Denial of Service' attack

Let’s check who owns these addresses (http://www.arin.net/):

```
OrgName:    Cable & Wireless
OrgID:      EXCW
Address:    3300 Regency Pkwy
City:       Cary
StateProv:  NC
PostalCode: 27511
Country:    US

NetRange:   216.32.0.0 - 216.35.255.255
CIDR:       216.32.0.0/14
NetName:    LEGACY-8
NetHandle:  NET-216-32-0-0-1
Parent:     NET-216-0-0-0-0
NetType:    Direct Allocation
NameServer: DNS01.EXODUS.NET
NameServer: DNS02.EXODUS.NET
NameServer: DNS03.EXODUS.NET
NameServer: DNS04.EXODUS.NET
Comment:    * Rwhois reassignment information for this block is 
available at:
Comment:    * rwhois.exodus.net 4321
Comment:    * For abuse please contact abuse@exodus.net
RegDate:    1998-07-30
Updated:    2002-10-30

…
```

Both IP addresses 216.33.87.10 and 209.67.29.9 belong to Cable & Wireless. Unfortunately, reverse lookup to get their hostnames wasn’t possible. This probably could have revealed more information about the purpose of these machines. Both IP addresses are not known to be well known attackers according to dshield.org (at least they are nor in their database).

Let's have a closer look at these packets

```
+------------+-------+------+--------+-------+-------+------------+
| ip_src     | ip_id | ttl  | ip_len | sport | dport | tcp_seq    |
+------------+-------+------+--------+-------+-------+------------+
| 3626063626 | 34525 |  242 |     64 |  2200 |    53 |  356149851 |
| 3626063626 | 29838 |  242 |     64 |  2202 |    53 | 1830859234 |
| 3626063626 | 18262 |  242 |     64 |  2201 |    53 | 1049394275 |
| 3510836489 | 10944 |  241 |     64 |  2100 |    53 | 1225823907 |
| 3510836489 |  7118 |  241 |     64 |  2101 |    53 | 1144910524 |
| 3510836489 | 47147 |  241 |     64 |  2102 |    53 | 1327763976 |
+------------+-------+------+--------+-------+-------+------------+
```

The TCP window size was 2048 on all packets, the IP length and destination port stays the same too. The IP ID is changing, as well as the TTL, source port and TCP sequence number. We are definitively not dealing with a tool which sends some crafted packets with constant values. As mentioned, the source port is changing, to be precise; it is always incrementing so these are no retries. We do have two sources for these attacks. What's interesting is the IP ID. Let's get this into another form

```
+---------------------+-------+-----------+
| timestamp           | ip_id | tcp_sport |
+---------------------+-------+-----------+
| 2002-08-27 20:14:34 | 34525 |      2200 |
| 2002-08-27 20:14:34 | 29838 |      2202 |
| 2002-08-27 20:14:34 | 18262 |      2201 |
| 2002-08-28 01:39:23 | 10944 |      2100 |
| 2002-08-28 01:39:23 |  7118 |      2101 |
| 2002-08-28 01:39:23 | 47147 |      2102 |
+---------------------+-------+-----------+
```

We receive three packets each time in approximately the same second. Each is coming from a different source port. So these are different connection attempts. Normally, the IP ID is incrementing by one each time a new IP packet is generated or they are chosen randomly. If this host is also following the "increment IP ID by one each time" rule, there are about 60000 IP packets sent between packet one and two. And all this in approximately the same second. We also have to consider, that we don't receive all packets in the original order. It is therefore possible that the original sending order was different (i.e. source port 2200 -> 2201 -> 2202) which would make sense. But even then, the IP IDs change really fast. But if the host is really sending that many packets a second which would result in this IP IDs, why is the source port number not also much higher on each connection attempt? If these IP IDs are for real, and not chosen randomly, than the machine generates/receives a lot of network traffic. I think it is more realistic, that the IP IDs are generated randomly. This again would not match our guess about Solaris being the operating system of our attacker (according to the TTL value). But probably we are dealing with an operating system with changed default settings.

But let's get back to our initial question about the goal of these packets.

Could it be an error?
It's possible that we are dealing with some bad implementation of a DNS client. But why would we have three connection attempts each time? And how does it come that we have so totally different IP IDs?

Some kind of padding?
Padding could be possible but it doesn’t look very realistic. Let's drop that one as it doesn't make much sense.

Hidden command for a tool?
The payload could be used as a certain pattern to trigger a response from a hidden tool. Using destination port 53 would also make sense here as a lot of firewall configurations allow traffic on this port for DNS zone transfers.

What about some kind of DoS attack?

A DoS attack could be possible. Let's check with google (http://www.goole.com) if there are some known vulnerabilities and exploits with a DNS service which could match here. Unfortunately, nothing specific could be found.

Some kind of reconnaissance?
I think reconnaissance is very likely. If the host is scanning a lot of other machines, even the IP IDs could be possible (I still tend to believe they are chosen randomly). Let's do some research on the Internet. According to some postings (see the Correlation section), different vendors of load balancers are known to send "strange" packets to do their measuring of round trip times. It looks like our traffic is fitting into this schema as well. By digging further, I'm quite sure we are dealing with some kind of round trip measurement. This would also explain why we are getting three packets every time.

Probability the Source Address was Spoofed

To measure the round trip times of certain packets to fine tune the load balancers, the originator of these probes would like to receive the result. Therefore he's not interested in spoofing the source addresses. Due to this fact, the probability the source address was spoofed is rated very low.

Description of the Attack

There seems to be always the same pattern: Three TCP connection attempts to port 53 within one second. The source port increments for each connection attempt. The attack is unusual as the SYN packets contain some payload and the DNS request is sent by TCP - normally most DNS operations are based on UDP. The payload in the SYN packet consists of zeroes only.

Attack Mechanism

Load balancers are using round trip time measurements to improve the load balancing decision. As soon as a user is requesting a certain webpage or from time to time, the load balancers are measuring round trip time to calculate their decisions. It is known that some commercial systems like F5's 3DNS or Cisco's Distributed Director may trigger intrusion detection alerts as the packets they are using aren't always "as normal as other network traffic". F5's 3DNS for example does exactly send three probes (mostly from source ports around 2100) to measure the round trip time.

Correlations

The attack schema seems to match a load-balancing product from F5 known as 3-DNS. For more details, visit their product page at http://www.f5.com/f5products/3dns/.
Some traces about possible alerts due to load balancers are mentioned in a presentation from Richard Bejtlich called "Interpreting Network Traffic" which can be found at http://www.parallaxresearch.com/dataclips/pub/infosec/IDS/first2000_IDS.ppt.
Other persons also reported similar events to the Snort mailing-list in December 2001 and January 2002 (http://www.mcabee.org/lists/snort-users/Dec-01/msg00613.html and http://archives.neohapsis.com/archives/snort/2002-01/0361.html).

Rinaldo Ribeiro has also detected similar events during his GCIA practical (http://www.giac.org/practical/Rinaldo_Ribeiro.doc).

Evidence of Active Targeting

This can definitively be regarded as active targeting - the load balancer is interested in our specific round trip time and is therefore sending a probe at our system.

Severity

The severity is calculated according to this formula:

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

Each element is worth 1 to 5 points, and the arithmetic gives us a range of -8 to +8.

Based on our analysis we are dealing with a load balancers round trip time measurement probe. This probe isn't an attack against our infrastructure. It's more an information gathering attempt to provide better load balancing and therefore better response times.

Criticality	As it seems, the probe is targeted against a user workstations	**2**
Lethality	This is just a probe for information gathering purposes	**1**
System	We don’t know anything about the patch level of the user’s system. At least we also had some virus warnings.	**3**
Network	Probably we have some firewall or router which is going to drop the packet later on, but this seems not realistic. The traffic isn’t illegal as such.	**2**

Severity would therefore be (2+1) - (3+2) = -2, which shows that we are dealing with a low severity incident. However, the detect is still interesting – the probes that are sent aren’t normal network traffic, they stick out. Knowing of the different load balancers signature would be of great help for an intrusion analyst to distinguish them from malicious traffic.

Defensive Recommendation

Checking the targeted host and applying the latest patches should be considered to achieve a good security level. The attack seems to belong to a "friendly" information gathering attempt but other activities can't be excluded completely. Checking the zone transfer settings for the used DNS servers should also be considered to insure that only intended servers can transfer zone information.

Allowing transfers for only certain IP’s can be achieved quite easily by inserting the corresponding option into the named.conf file of Bind.

```
zone "test.com" in {
        type master;
        file "test.com.zone";
        allow-transfer {
                1.2.3.4;
        };
};
```

The border firewall or router should be further improved. We can't say for sure how the network setup looks like, but if CISCO-Dev#1 is the border router/firewall, the ingres filtering isn't adequate.

Multiple Choice Test Question

What is not quite "normal" with the following packet?

```
[**] [1:526:4] BAD TRAFFIC XXXXX [**]
[Classification: Misc activity] [Priority: 3]
08/27-19:14:34.524488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33
type:0x800 len:0x4E
216.33.87.10:2200 -> 138.97.18.88:53 TCP TTL:242 TOS:0x0
ID:34525 IpLen:20 DgmLen:64
******S* Seq: 0x153A6A5B  Ack: 0x0  Win: 0x800  TcpLen: 20
```

a) The datagram length field holds a wrong value
b) This is a SYN packet with payload
c) The TTL value is way too high
d) Everything is fine and "normal"

Normally, SYN packets doesn't come with a payload, so answer (b) is correct. Answer (a) is not correct as the datagram length is correct. TTL values of 242 are very well possible. Solaris for example uses a default TTL value of 255 which would result on a distance of 13 hops which seems reasonable.

Posting to incidents.org

I also posted this detect on the 03/10 to the intrusions@incidents.org mailing list to get some feedback and opinions on my detect. The original posting can be found at http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00132.html.
Andre Rucker Jones did post some comments to rephrase or restructure certain parts in the detect – thanks a lot for your feedback. Unfortunately, other feedback wasn't received for this detect.

References

Anonymous. "Default TTL Values in TCP/IP". Switch. 2003
URL: http://secfr.nerim.net/docs/fingerprint/en/ttl_default.html (March 11 2003)

Ribeiro, Rinaldo. "GCIA Practical Exam – Detect #1". 2003
URL: http://www.giac.org/practical/Rinaldo_Ribeiro.doc (March 10 2003)

Stevens, W. Richard. "TCP/IP llustrated, Volume I – The Protocols". Addison Wesley, 1994

Roesch and Green. “Snort User’s Manual Chapter 2: Writing Snort Rules”. 2003
URL: http://www.snort.org/docs/writing_rules/chap2.html (March 10 2003)

Detect #3: Bad Traffic UDP port 0

Log Trace

For detect #4 I will focus on the following detect of the snort-log file:

```
[**] [1:525:4] BAD TRAFFIC udp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
11/11-19:29:54.796507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5C
211.194.68.39:1025 -> 207.166.72.218:0 UDP TTL:109 TOS:0x0 ID:2062
IpLen:20 DgmLen:78
Len: 137
[Xref => nessus 10074][Xref => cve CVE-1999-0675]
```

Source of Trace

The network traffic for this detect was obtained from the incidents.org website, namely http://www.incidents.org/logs/Raw/2002.10.11.

This file is a "real world" binary log generated by snort and sanitized for use within the GCIA practical. What need to be known for this log file:

- Binary mode logging
- Only the packet that violated a rule is in this log
- Unknown set of Snort rules
- IP address of the protected network is sanitized.
- Checksums are modified
- No ICMP, DNS, SMTP or web traffic.
- IP addresses belonging to non-local hosts are the real ones

For more details have a look at http://www.incidents.org/logs/Raw/README

Detect was Generated by

I used Snort version 1.9.1 (latest version at the time of writing) with the standard 1.9.1 ruleset. Setting the network information variables was achieved by supplying the corresponding variables by command line option

```
$ snort -c /etc/snort/snort.conf -d -e -l snort-logs/ -r
2002.10.11 -S HOME_NET=207.166.0.0/16 -S
EXTERNAL_NET=\!207.166.0.0/16
```

```
Log directory = snort-logs/
TCPDUMP file reading mode.
Reading network traffic from "2002.10.11" file.
snaplen = 1514

        --== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file /etc/snort/snort.conf
--- SNIP ---
```

```
1310 Snort rules read...
1310 Option Chains linked into 148 Chain Headers
0 Dynamic rules
+++++++++++++++++++++++++++++++++++++++++++++++++++

Rule application order: ->activation->dynamic->alert->pass->log

        --== Initialization Complete ==--

-*> Snort! <*-
Version 1.9.1 (Build 231)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
Run time for packet processing was 31.93486 seconds


=====================================================================

Snort processed 11116 packets.
Breakdown by protocol:                Action Stats:

    TCP: 11103      (99.883%)         ALERTS: 7738
    UDP: 1          (0.009%)          LOGGED: 7738
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 11         (0.099%)
=====================================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0          (0.000%)
    Control Packets:    0          (0.000%)
    Data Packets:       0          (0.000%)
=====================================================================
Fragmentation Stats:
Fragmented IP Packets: 16         (0.144%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
=====================================================================

TCP Stream Reassembly Stats:
   TCP Packets Used:      0          (0.000%)
   Reconstructed Packets: 0          (0.000%)
   Streams Reconstructed: 0
=====================================================================
```

Snort tells us that we mostly have TCP packets (99.9%), one UDP packet and 16 fragmented packets. Having a look at the alerts file shows quite an amount of alerts, most of them seem to be possible SCANs.

By telling Snort to log into a MySQL database, getting some more information about the number of alerts and other statistics is quite easy. I extracted to information about the distribution of the alerts, therefore which alerts we had and how often.

```
+--------------------------------+------+
| sig_name                       | anz  |
+--------------------------------+------+
| BAD TRAFFIC udp port 0 traffic |    1 |
| MISC Tiny Fragments            |    1 |
| SCAN SOCKS Proxy attempt       |    3 |
| BAD TRAFFIC same SRC/DST       |   11 |
| BAD TRAFFIC bad frag bits      |   15 |
| BAD TRAFFIC tcp port 0 traffic |   29 |
| SCAN nmap TCP                  |   46 |
| SCAN Proxy (8080) attempt      | 3788 |
| SCAN Squid Proxy attempt       | 3844 |
+--------------------------------+------+
```

For detect #3 I'd like to focus on the 'BAD TRAFFIC UDP port 0 traffic' alert, which was detected once and looks the following way

```
[**] [1:525:4] BAD TRAFFIC udp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
11/11-19:29:54.796507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5C
211.194.68.39:1025 -> 207.166.72.218:0 UDP TTL:109 TOS:0x0 ID:2062
IpLen:20 DgmLen:78
Len: 137
[Xref => nessus 10074][Xref => cve CVE-1999-0675]
```

The signature which triggered this alert is located in the bad-traffic.rules file:

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

The alert is setup to detect UDP packets designated or originated from port 0 from our internal network. There are also some references listed, i.e. CVE-1999-0675 which states (http://cve.mitre.org/cve/) "Check Point FireWall-1 can be subjected to a denial of service via UDP packets that are sent through VPN-1 to port 0 of a host." It sounds as if this attack was targeted at a CheckPoint FireWall-1 that this could have been bad.

What does the alert tell us?

- Packet was capture on the 11th November (which seems to be a little bit strange as the log file is called 2002.10.11 but let's ignore this)
- Ether type was IP (type:0x800)
- Frame length is 92 (0x5c)
- UDP packet
- Source 211.194.68.39, port 1025
- Destination 207.166.72.218, port 0
- Time to live (TTL) is 109
- IP ID is 2062
- Datagram length is 78 (IP Header + UDP Header + data), which tells us that the payload has to be 78 - 20 (IP Header, no options) - 8 (UDP Header) = 50
- The alert is called 'BAD TRAFFIC udp port 0 traffic'

With this data we are going to dig up the packet (snort_logs/211.194.68.39)

```
[**] BAD TRAFFIC udp port 0 traffic [**]
11/11-19:29:54.796507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5C
```

```
211.194.68.39:1025 -> 207.166.72.218:0 UDP TTL:109 TOS:0x0 ID:2062
IpLen:20 DgmLen:78
Len: 137
D5 86 01 00 00 10 00 01 00 00 00 00 00 00 20 43  .............. C
4B 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  KAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 00  AAAAAAAAAAAAAAA.
00 21                                            .!
```

Or the tcpdump hex output

```
19:29:54.796507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 92: 211.194.68.39.1025
> 207.166.72.218.0: udp 129
                         4500 004e 080e 0000 6d11 5f71 d3c2 4427
                         cfa6 48da 0401 0000 0089 003a d586 0100
                         0010 0001 0000 0000 0000 2043 4b41 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4100 0021
```

Is this normal UPD payload?
This payload looks as if we are dealing with a NetBIOS wildcard search which is used to query a host for its NetBIOS table by sending a wildcard '*' instead of a hostname. A NetBIOS wildcard lookup must contain the string "CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" followed by 0x0000 as payload to match the wildcard.

Probability the Source Address was Spoofed

Let's check for packet crafting. A TTL of 109 seems to be a reasonable value (could have been a Windows NT 4.0 [default TTL 128] which is then 19 hops away).
The logs also only show this packet coming from that source address and none heading for this address. Checking this address with dshield (http://www.dshield.org) does also not reveal some new information.
What about the destination port 0?
I checked with google (http://www.google.com) to see if there is more information about using port 0 in UDP communication. Several sources were found, most of them came to more or less the same conclusion: Technically it is possible to use port 0 in IP communication, but it doesn't make much sense as "no know" applications use it. It is known that some hacker tools may use port 0 and in UNIX network programming, a source port of 0 is used to assign the next free port number to a socket. But does this mean the packet was crafted? Not necessarily as it is technically possible and not even forbidden to use the port number 0.

By looking closer at the packet, more strange symptoms can be found. Having a closer look at the different length information looks promising:

- IP header tells us that we have a length of 78
- UDP length tells us something about 137
- If we subtract the IP header length (20 bytes) from the value of the IP total length field leaves us with 58 bytes for UDP, which will leave us 50 bytes for DUP payload
  ➔ So we have IP telling us 50 bytes payload and UPD telling us something

about 137 (total length) - 8 (UDP Header) = 129 bytes. Seems like this packet is crafted or we have some kind of error somewhere.

An attack against a CheckPoint firewall doesn't seem to be reasonable. The vulnerability is already quite old and most (if not all) systems should be patched. The attack also only works in conjunction with a VPN – the probability that this is really an attack against a CheckPoint firewall seems very low.

When taking the TCP/IP bible by hand ("TCP/IP Illustrated, Volume 1" by W. Richard Stevens) and looking up some TCP/IP fields and counting some bits it strikes me. Could it be that we're really dealing with some sort of network error?

The original packet we received could have some missing or inserted fields. Let's change the original packet we received

```
19:29:54.796507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 92: 211.194.68.39.1025
> 207.166.72.218.0: udp 129
                         4500 004e 080e 0000 6d11 5f71 d3c2 4427
                         cfa6 48da 0401 0000 0089 003a d586 0100
                         0010 0001 0000 0000 0000 2043 4b41 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4100 0021
```

to the following

```
                         4500 004e 080e 0000 6d11 5f71 d3c2 4427
                         cfa6 48da 0401 0089 003a d586 0100 0010
                         0001 0000 0000 0000 2043 4b41 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4100 0021 0001
```

We deleted some data and appended some new values. If we are going through the packet now, we have the following information:

- IP header
  20 (0x5 * 8) bytes long
  78 (0x004e) bytes total length
  2062 (0x080e) IP ID
  109 (0x6d) TTL
  IP source 211.194.68.39
  IP destination 207.166.72.218
- UDP header
  1025 (0x0401) source port number
  137 (0x0089) destination port number
  58 (0x003a) UDP length
- Payload
  Well known pattern for a NetBIOS wildcard scan

Suddenly, even the different length field match (total length – TCP header – UDP header = payload). This new view on the problem leads to the conclusion that the source IP address is not spoofed as the attacker would like to see an answer and is especially interested in what the answer might be.

Description of the Attack

We know that we are having a NetBIOS wildcard lookup hidden in a strange looking packet. What we don't know if is this happened on purpose to hide the real meaning of the packet, to evade a possible IDS or if there simply happened an error on the way the packet traveled. It doesn't make much sense to hide a packet on purpose in such a way as the receiving host wouldn't know anything about the "real" meaning. The probability that we are dealing with some sort of network error which generated this anomaly is much higher. It would be interesting to have the original checksums ready as they should reveal more information.

Attack Mechanism

It is possible that we are dealing with some kind of network error which resulted in changing the packet's content, in this case inserted some new bytes into the packet. As the newly inserted bytes only affect the UDP header and therefore are not part of the IP header checksum, the error would not have been detectable by the router the packet passed by as they are only checking and updating the IP header checksums.

Correlations

This particular detect has never been seen before. As this is more a network problem as a real attack this doesn't surprise.

Evidence of Active Targeting

The logs don't show traces of active targeting. The attacked system as well as the attacker doesn't show up in association with other alerts. It could certainly be that the host was probed a few days or week earlier. But this seems to be a random "probe".

Severity

The severity is calculated according to this formula:

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

Each element is worth 1 to 5 points, and the arithmetic gives us a range of -8 to +8.

Criticality	We don't know the target system but the probability is high that this is a Windows or Unix machine providing NetBIOS services (e.g. file sharing). But as long as we don't know for sure let's assume the worst case.	**5**
Lethality	This kind of attack seems very unlikely to succeed	**1**
System	Again, we don't have enough information, so let's go with a fairly bad case scenario (older operating system, patches missing)	**3**
Network	As far as we can say, there were no network countermeasures so far (the packet reached the Snort sensor). Let's stick to a worst case scenario.	**1**

Severity would therefore be (5+1) - (3+1) = 2 which is moderate. Having an eye on the targeted machine as well as the attackers IP should be a wise approach.

Defensive Recommendation

To be sure not to have such “invalid” packets coming into your network and probably having some unknown effects it is best to drop them at the perimeter. There is no sense in letting such packets pass the perimeter. Adjusting the firewall rules in a way to drop such packets is recommended. If similar errors are occurring quite often, checking the network cabling and the network hardware could also solve the problem as there could be possible problems which lead to such anomalies.
Checking the targeted machine and installing the latest patches should further increase the network security.

Multiple Choice Test Question

What triggered the "BAD TRAFFIC" alert?

```
[**] [1:525:4] BAD TRAFFIC xxxxxxxxxxxx [**]
[Classification: Misc activity] [Priority: 3]
11/11-19:29:54.796507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5C
211.194.68.39:1025 -> 207.166.72.218:0 UDP TTL:109 TOS:0x0 ID:2062
IpLen:20 DgmLen:78 Len: 137
[Xref => nessus 10074][Xref => cve CVE-1999-0675]
```

a) IP Length is too small
b) Destination TCP port 0 is invalid
c) Destination UDP port 0 is normally not used
d) There is nothing wrong with this packet, Snort must be mistaken

Answer is (c) as according to the RFC768 a port number of 0 is technically not invalid, but this port number is not used as a destination port in normal network traffic as there are no services are listening on that port (port 0 is reserved/restricted).

Posting to incidents.org

On first posting to incidents.org I thought that we are dealing with some sort of reconnaissance or denial of service attack. Ronny Rietveld then mentioned the possibility that this could be a “simple” error somewhere. He pointed at the packet information and the coincidence, when leaving out some bits of data, the packet looks quite normal. I think his idea and detect made much more sense than my first thought about some sort of attack – So I modified by detect accordingly. The original posting of Ronny can be found at http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00087.html.

References

Stevens, W. Richard. “TCP/IP llustrated, Volume I – The Protocols”. Addison Wesley, 1994

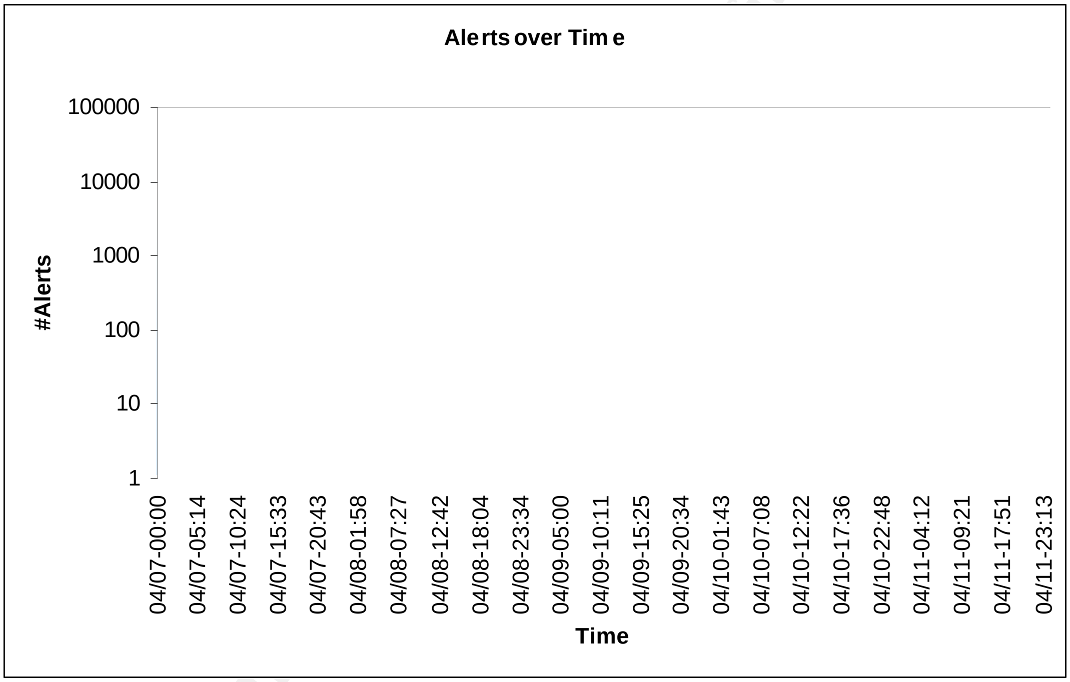
Roesch and Green. “Snort User’s Manual Chapter 2: Writing Snort Rules”. 2003
URL: http://www.snort.org/docs/writing_rules/chap2.html (March 21 2003)

Assignment #3: Analyze This

This is a scenario-based assignment. You have been asked to provide a security audit for a University by analyzing logs from their intrusion detection system to produce an analysis report. You should be especially alert for signs of compromised systems or other network problems.

Executive Summary

Intrusion Detection logs were analyzed over a period of five days, showing 430'867 alerts and 112576 scans. The graph "Alerts over Time" shows the number of Events of Interest over these five days. Several hundred events were reported each hour, with an enormous peak around April 7th, 2003 7 o'clock where a denial of service attack was conducted from the University network to a target in Europe.



This high number of alerts seems frightening – are there so many attacks launched or is the University full of hackers who launch attacks against others?

During my analysis, multiple potential hackers and systems which have been hacked could be identified. But this is only a small number compared to these thousands of events we've recorded. They are an result of very unspecific intrusion detection signatures which trigger far to often on unsuspicious traffic. On the other hand, the University network is poorly defended against intruders as nearly no perimeter security is in place. Several corrective actions should be set up immediately to fix serious problems and enhance the overall network security. A list of recommendations is provided in the chapter "Conclusion and Recommendation".

The Log Files and Data Preparation

The University provided me with three sets of log files, covering the period of April 7 through April 11, 2003. These log files represent the results of 5 days of network traffic analysis. The logs were generated by a Snort IDS system with a standard rule base. The alert log files which were used are:

Filename	Size (KB)
alert.030407	28’077
alert.030408	4’318
alert.030409	7’157
alert.030410	6’972
alert.030411	7’578

For better analysis and overview, all these log files were scanned and inserted into a database. The different attributes like source and destination IP as well as source and destination port were separated in corresponding fields. With this setup, latter analysis with SQL queries were made possible and much easier than dealing with these huge log files.

For the same days, the information about scans was provided in separate files. These were:

Filename	Size (KB)
scans.030407	3’408
scans.030408	1’779
scans.030409	3’192
scans.030410	1’678
scans.030411	10’008

For easier analysis, the information about individual scans was also inserted into a relational database.

Log files with out of spec packets, that are TCP packets with strange or invalid flags set, were also provided. These packets are all involved in events which were generated in the first two set of logs. The files were:

Filename	Size (KB)
oos_report_2003_04_07_31826	791
oos_report_2003_04_08_2317	571
oos_report_2003_04_09_32618	701
oos_report_2003_04_10_10565	5’921
oos_report_2003_04_11_11835	4’186

Most Frequent Events

In these five days, more than 500’000 alerts and scans were recorded which results in an average of more than 11 events every second for each day. Which such a high number of alerts the question arises if we are dealing with a bunch of potential criminals which are attacking other networks or if the University is attacked very often. Another very likely possibility could be the “not so good” configured IDS. Probably a lot of false positives are generated.

To be able to draw some conclusions here, the alerts that were triggered more than 10’000 times are analyzed in more detail.

Alert	Count
TCP SRC and DST outside network	181’671
SMB Name Wildcard	112’665
Watchlist 000220 IL-ISDNNET-990517	33’090
CS WEBSERVER – external traffic	19’814
High port 65535 udp – possible Red Worm – traffic	17’962
spp_http_decode: IIS Unicode attack detected	16’243

Frequent Alert Details: TCP SRC and DST outside network

Reported 181’671 times, severity medium
Standard Snort Signature ID: None

```
04/07-00:45:06.232206  [**] TCP SRC and DST outside network [**]
192.168.2.100:4375 -> 213.35.101.25:80
04/07-00:14:52.971968  [**] TCP SRC and DST outside network [**]
192.168.1.100:4662 -> 24.243.16.98:14174
04/07-00:35:48.047070  [**] TCP SRC and DST outside network [**]
0.0.0.0:5501 -> 65.68.101.82:4729
04/07-03:21:11.565795  [**] TCP SRC and DST outside network [**]
192.168.1.101:2677 -> 65.49.92.75:2908
04/07-04:45:56.479569  [**] TCP SRC and DST outside network [**]
158.64.156.146:1874 -> 216.152.64.158:6667
04/07-04:45:56.479734  [**] TCP SRC and DST outside network [**]
158.64.156.147:1524 -> 216.152.64.158:6667
```

Analysis: The source IP address for these alerts seems to be very random. We have about 1% of all alerts generated by source IP 192.168.1.100. All other 181’000 alerts are generated by different IP’s. Even by looking at the different attacking networks, no accumulation of alerts for a certain network can be found. The same analysis was done for the destination address with revealed more as there is a certain clustering.

Destination IP	Alerts triggered	Percent
62.13.46.133	49396	27.2%
217.75.98.40	41195	22.7%
216.152.64.213	27536	15.2%
216.152.64.143	26837	14.8%
216.152.64.158	24214	13.3%
194.23.46.226	10695	5.9%
Others	1798	0.9%

It’s interesting to see that more than 40% of the alerts were triggered in conjunction with the subnet 216.152.64.0/24.

A whois query for the IP address 62.13.46.133 shows us that the owner of this IP is a certain "Gate Company SA" in Stockholm, Sweden which seems to be an IT consulting and services company.

```
% 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:      62.13.46.0 - 62.13.46.255
netname:      GATECOMPANY-SE-NET
descr:        Gate Company AB
descr:        Stockholm
country:      SE
admin-c:      ME4036-RIPE
tech-c:       ME4036-RIPE
status:       ASSIGNED PA
mnt-by:       AS8434-MNT
changed:      kandra@utfors.se 20010724
source:       RIPE

route:        62.13.0.0/17
descr:        UTFORS-BLK
origin:       AS8434
member-of:    AS8434:RS-PA-BLK
remarks:      *** Contact abuse@utfors.se regarding ABUSE please! ***
mnt-by:       UTFORS-MNT
changed:      hakan@utfors.net 20020424
changed:      hakan@utfors.net 20030203
source:       RIPE

person:       Mats Eriksson
address:      Gate Company AB
address:      Box 709, Oxtorget 3
address:      SE-10387 Stockholm
address:      Sweden
e-mail:       mats.eriksson@gatecompany.com
phone:        +46 8 221090
fax-no:       +46 8 232480
mnt-by:       AS8434-MNT
nic-hdl:      ME4036-RIPE
changed:      kandra@utfors.se 20010724
source:       RIPE
```

The other top scorer, IP 217.75.98.40 is also registered to a company from Sweden called "Port 80". It seems that Port 80 is an ISP. The three addresses from the 216.152.64.0/24 block belong to a company called "Webmaster Inc." who offers messaging servers for their customers. The last IP, 194.23.46.226, belongs to Telia SA, a leading telecommunications group in the Nordic and Baltic regions.

Looking at the affected ports shows nothing special on the source port. The destination port although shows that 56% are destined at port 80 and 43% at port 6667. All alerts were port 6667 was involved belong to the 216.152.64.0/24 block. Port 6667 is often used for IRC (Internet Relay Chat), an Internet chat protocol based on TCP which is widely used in the warez and hacker communities. It doesn't seem to make sense to use a wrong source IP address in a chat session besides the origin wanted to spoof their source address. Is it possible that a denial of service attack was launched?

All alerts were triggered between 04:37 and 04:53 on April 7. In certain minutes more than 23'000 alerts were generated, which means we have about 380 alerts each second. If these are all new connection attempts, we are certainly dealing with some sort of denial of service, especially if all source addresses are spoofed and different (which is certainly the case here). It's also interesting to see that the source IP addresses are incremented in a logical pattern. There is certainly a tool at work which attempts some sort of denial of service attack. The same is happening to the other targeted hosts, but this time port 80 is targeted.

Correlation: Denial of service attacks are well known. A lot of tools exist to do this very easily. Although, publications concerning this exact attack were not found on the Internet.

Conclusion: As we are dealing certainly with some denial of service attacks this should be investigated further. The attacker is using a lot of bandwidth and he is using the Universities network for illegal activities. The first action should be to drop outbound traffic which has not a source address from local IP range. In this way, spoofed IP's can no longer escape the University's network.
In a second attempt, the bad guy should be tracked down by using the associated MAC address of these alerts (hopefully he's not faking this as well) or by logging detailed information on most routers to further track him down.

Frequent Alert Details: SMB Name Wildcard

Reported 112'665 times, severity high
Standard Snort Signature ID: None

```
04/07-00:00:02.763936  [**] SMB Name Wildcard [**] 208.11.160.61:137
-> MY.NET.222.206:137
04/07-00:00:02.932505  [**] SMB Name Wildcard [**] 4.61.236.24:1077 -
> MY.NET.165.181:137
```

Analysis: These alerts are describing normal NetBIOS name resolution traffic as it was also reported by others. These alerts wouldn't trouble us much as long as the source and destination IP's are within our own network. But querying the database reveals that 100% of these alerts were triggered with non local source addresses and local destination addresses (the signature is most probably set up to only trigger on incoming traffic, which makes perfectly sense). This should certainly ring some bells. Why should we allow incoming NetBIOS traffic to port 137? Is the University allowing to access Windows shares from external? Or is there a common University policy which says "Allow everything"?
Further looking at these alerts shows that the sources for the "SMB Name Wildcard" are a lot of IP addresses all around the globe. The same situation can be found with the target IP's – All are within our own MY.NET network but no specific host is attacked significantly more often than others. So it seems that these are random scans from external machines.

Correlation: SMB Name Wildcard alerts are well known and reported by several people and organizations. Tod Beardsley's GCIA practical is mentioning the same kind of alert, although he mostly had internal traffic. Bryce Alexander's Intrusion FAQ Port 137 Scan deals with these attacks quite extensively. Other sources can be found on the Internet, including information from CERT.

Conclusion: Allowing incoming traffic for port 137 will open up a lot of potential to attack Windows machines and/or get the names for available Windows shares. According to CERT Incident Note IN-2000-02, traffic to port 137-139 should be blocked to hinder access to Windows shares to the public.

Frequent Alert Details: Watchlist 000220 IL-ISDNNET-990517

Reported 33’090 times, severity unknown
Standard Snort Signature ID: None

```
     04/08-00:22:08.293759  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
     212.179.66.17:80 -> MY.NET.252.122:4854
     04/08-00:22:08.671022  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
     212.179.66.17:80 -> MY.NET.252.122:4854
```

Analysis: This seems to be a rule which triggers on certain interesting traffic. *“This is a custom alert created on May 17, 1999 no doubt to watch for malicious activity from the Israeli ISP Bezeq International. There must have been a history of security issues with this network in the past.”* – Source Les Gordon, GCIA practical.
A whois lookup for “IL-ISDNNET-990517” reveals that this is a network in Israel which seems to be watched.

```
% 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:      212.179.0.0 - 212.179.255.255
netname:      IL-ISDNNET-990517
descr:        PROVIDER
descr:        ISDNet LTD
country:      IL
admin-c:      MR916-RIPE
tech-c:       ZV140-RIPE
status:       ALLOCATED PA
notify:       hostmaster@bezeqint.net
mnt-by:       RIPE-NCC-HM-MNT
mnt-lower:    AS8551-MNT
mnt-routes:   AS8551-MNT
changed:      hostmaster@ripe.net 19990517
changed:      hostmaster@ripe.net 20020912
changed:      hostmaster@ripe.net 20020926
source:       RIPE
person:       Miri Roaky
address:      bezeq-international
address:      40 hashacham
address:      petach tikva 49170 Israel
phone:        +972 1 800800110
fax-no:       +972 3 9203033
e-mail:       hostmaster@bezeqint.net
mnt-by:       AS8551-MNT
nic-hdl:      MR916-RIPE
changed:      hostmaster@bezeqint.net 20021027
```

```
changed:       hostmaster@bezeqint.net 20030204
source:        RIPE
person:        Zehavit Vigder
address:       bezeq-international
address:       40 hashacham
address:       petach tikva 49170 Israel
phone:         +972 1 800800110
fax-no:        +972 3 9203033
e-mail:        hostmaster@bezeqint.net
mnt-by:        AS8551-MNT
nic-hdl:       ZV140-RIPE
changed:       hostmaster@bezeqint.net 20021027
changed:       hostmaster@bezeqint.net 20030204
source:        RIPE
```

The associated ports to this alert are 3933 (30%) and 1214 (20%). Port 1214 is mostly used for KaZaA, a peer-to-peer file sharing program, often used to share MP3 or warez.

Correlation: Les Gordon mentioned this alert in his GCIA practical. Other correlations to this alert were found on the Internet but without much information.

Conclusion: The intent for this alert is not clearly known, but should be checked. If the university has a policy which doesn't allow peer-to-peer file sharing, the users violating this rule should be warned.

Frequent Alert Details: CS WEBSERVER – external traffic

Reported 19'814 times, severity unknown
Standard Snort Signature ID: None

```
04/08-01:00:18.238331  [**] CS WEBSERVER - external web traffic [**]
130.194.13.180:21666 -> MY.NET.100.165:80
04/08-00:27:36.279611  [**] CS WEBSERVER - external web traffic [**]
68.51.166.255:1159 -> MY.NET.100.165:80
```

Analysis: About 99.9% of all this traffic was destined at IP MY.NET.100.165 port 80 (HTTP). The alert's name allows the assumption, that MY.NET.100.165 is a web server with no intended external traffic or with external traffic which should be watched. The top external talker to this web server seems to be 216.39.48.2 (45%). Querying the database shows, that MY.NET.100.165 is involved with a lot of alerts. Please see chapter 'Events of Interest' for a more in-depth analysis of this system.

Correlation: Some clues regarding this alert can be found on google, but nothing seemed worth mentioning here.

Conclusion: This is obviously a web server which is talking to external browsers. This alert has to be re-considered. If external traffic from this web server is not allowed, this policy should be enforced by a corresponding firewall rule set. If this alert is only placed to "measure" traffic to this web server, I would disable this rule and find other methods of measuring the traffic.

Frequent Alert Details: High port 65535 udp – possible Red Worm – traffic

Reported 17'962 times, severity noise
Standard Snort Signature ID: None

```
04/08-00:16:37.527333  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.201.58:65535 -> 213.10.166.78:5121
04/08-00:16:38.024304  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.201.58:65535 -> 80.8.2.139:5121
```

Analysis: MY.NET.201.58 was involved 7953 times (45%) as target when this alert was triggered. IP 66.42.68.210 was involved as target for another 4588 times (25%). The missing 30% are distributed over a wide range of IP addresses. MY.NET.201.58 is acting as source for this alert for more than 8400 times (47%), 66.42.68.210 is responsible for about 26% and 4.46.32.83 for 7%. A fast query to the database reveals that both IP addresses are only involved in this kind of alerts (MY.NET.201.58 also has triggered two SMB Name Wildcard alerts). Further analysis shows, when MY.NET.201.58 is acting as source the destination port is mostly 5121 (95%) or 4121, 5125 or 5122. Port 5121/UDP is used for “Neverwinter Nights” (a role playing game) which can be played over the Internet. It is possible that this game was really played here.
The most interesting fact is probably that nearly 100% of all times the IP 66.42.68.210 is listed as source for this alert, MY.NET.201.58 is the target and vice versa. This external IP belongs to a company called “Pac West Telecomm Inc.” which seems to be a telephone and Internet company. The alert is also spread through the whole time period popping up quite regularly every hour a few times (between 2 and 900 times).

According to the data we have here, I would say we are dealing with a lot of IDS noise – Red Worm does normally use TCP for it's traffic an not UDP which decreases the likelihood we're dealing with a real Red Worm incident. The Adore Worm would also use port 65535 for its traffic, but also TCP and not UDP which was used here. As most of these alerts are generated by two involved IP's (which are obviously talking to each) other I would guess this is more normal traffic which triggered this alert signature.

Correlation: Similar alerts have been investigated by some GCIA students, including Les Gordon.

Conclusion: The signature for this specific alert should be adapted – It seems as if this signature would trigger a lot of false alerts.

Frequent Alert Details: spp_http_decode: IIS Unicode attack detected

Reported 16'243 times, severity medium
Standard Snort Signature ID: http_decode

```
04/07-00:01:22.636391  [**] spp_http_decode: IIS Unicode attack
detected [**] MY.NET.98.141:4264 -> 211.233.53.216:80
04/07-00:01:22.636391  [**] spp_http_decode: IIS Unicode attack
detected [**] MY.NET.98.141:4264 -> 211.233.53.216:80
```

Analysis: Unicode attacks are mostly used to escape the default web server's root directory and get access to files outside the normally served directories. Code Red, Code Red II and Nimda all rely on some sort of Unicode attack to get access to files by

providing specially built relative directories – Unicode is used to hide the request for a certain file so that the web server will not disallow the access. The http_decode Snort preprocessor is specifically designed to watch for Unicode-encoded “\” “/” and “.” characters on common HTTP ports.
This alert is triggered based on certain content within the payload. Unfortunately, this payload can occur a number of times in data transfer traffic (like images, files, etc.) which will result in a high false positive rate.

Correlation: This alert is well known and discussed in several papers, including the GCIA practical from Tod Beardsley and Les Gordon. Tod Beardsley mentions that this alert always signifies an attack – I don’t think this is perfectly right as other traffic can contain these patterns.

Conclusion: Don’t rely on IDS so reactively act on Code Red attacks. It’s better to establish some egres and ingress content filtering to get rid of this sort of traffic.

Most Frequent Scans

The following table lists the the “top scanners”, therefore the IP’s which were recorded most often for scanning one or more systems. Looking through the log files, it seems as if the scan files for the used period were not obfuscated as no MY.NET IP addresses are used. According to the number of alerts on certain subnets and a whois query for 130.85.98.176 makes me believe that the 130.85.0.0/16 is considered to be the MY.NET.0.0/16 network as it also belongs to a University.

```
OrgName:    University of Maryland Baltimore County
OrgID:      UMBC
Address:    UMBC University Computing
City:       Baltimore
StateProv:  MD
PostalCode: 21250
Country:    US

NetRange:   130.85.0.0 - 130.85.255.255
CIDR:       130.85.0.0/16
NetName:    UMBCNET
NetHandle:  NET-130-85-0-0-1
Parent:     NET-130-0-0-0-0
NetType:    Direct Assignment
NameServer: UMBC5.UMBC.EDU
NameServer: UMBC4.UMBC.EDU
NameServer: UMBC3.UMBC.EDU
Comment:
RegDate:    1988-07-05
Updated:    2000-03-17

TechHandle: JJS41-ARIN
TechName:   Suess, John J.
TechPhone:  +1-410-455-2582
TechEmail:  jack@umbc.edu

# ARIN WHOIS database, last updated 2003-05-01 20:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

All IP addresses which were seen to be from the 130.85.0.0/16 network have been replaced by MY.NET.x.x to be consistent with the other log files.

Source IP	Number of scans
80.14.81.238	8100
MY.NET.70.136	5794
129.44.184.64	5472
157.181.237.50	5251
MY.NET.98.176	4248
MY.NET.97.154	3990
MY.NET.1.3	3750
66.183.227.87	3516

The top eight most scanned destination addresses

IP	Number of scans
MY.NET.225.174	5473
MY.NET.236.146	3517
80.139.178.30	498
80.200.118.241	498
204.183.84.240	496
4.63.63.210	488
217.229.198.234	486
80.126.148.112	486
213.84.36.25	484

How was scanned? What kinds of scans were used most often? The following table lists scan types used more often than 1000 times

Scan Type	% of all scans	Number of times used
UDP	50.7%	57097
SYN#******	35.3%	39716
NULL#*****	6.5%	7288
FIN#******	2.5%	2804
NOACK#**U*	1%	1113
INVALIDACK	0.9%	1051
SYN#12****	0.9%	1033

UDP scans are the most frequent (more than 50%), followed by SYN scans (also known as half-open scans) which make about 35%. The NULL scans, where no TCP flags are set, have been seen about 7200 times. FIN scan are also seen a number of times. All other scans are relatively rare (compared to the others) and use mostly different combinations of TCP flags.

It's interesting to notice that we have no internal scans listed in the logs. Not one scan can be found, where the source and target IP are within the MY.NET.0.0/16 network. I assume that the signatures are not configured to trigger on these scans or the Snort sensor wasn't in the right place to see them as it is quite impossible not to have internal scans.

Our top two scanners (80.14.81.238 and MY.NET.70.136) as well as the top scan target (MY.NET.225.174) are further investigated in the chapter ‘Interesting Hosts’.

Events of Interest

This chapter will show the most interesting alerts and investigate what they are about, what happened and what should be done.

Alert: DDOS shaft synflood incoming

Reported 860 times, severity medium

Analysis: *“This signature looks for SYN packets with a particular sequence number.”* – source Ryan Russel. Accoring to Ryan’s statement, this kind of alert is triggered very often in normal traffic as each sequence number can occur from time to time. But the case here seems different: All 860 alerts where generated in about 60 seconds and all attacks came from one source IP address - 206.167.20.2. This IP is registered for an organization called ‘Regroupement Loisir Quebec’ which also acts as some sort of ISP.
Checking the logs shows that this specific IP was only doing his DDOS shaft synflooding.

Conclusion: The attacked IP’s as well as the attackers IP (probably better subnet) should be added to a watch list.

Alert: DDOS mstream client to handler

Reported 77 times, severity high

Analysis: MY.NET.105.48 was targeted 74 times, MY.NET.205.174 two times both by a lot of different IP’s. The mstream is a distributed denial of service attack tool which communicates with it’s agents to launch coordinated attacks against certain IP’s. Nearly all alerts were triggered with a destination port 15104. It looks as if these two IP’s could be DDoS masters commanding other agents. MY.NET.105.48 is indeed a very suspicious host as a lot of other alerts had been triggered with this IP involved. The system will be further investigated in the ‘Interesting Hosts’ chapter.

Conclusion: Both hosts should be checked to assure they are not part of a distributed denial of service agent network (or even the commander of one).

Alert: Possible trojan server activity

Reported 799 times, severity medium

```
04/07-06:16:58.246684  [**] Possible trojan server activity [**]
200.60.137.73:27374 -> MY.NET.206.102:1995
04/07-06:16:58.249887  [**] Possible trojan server activity [**]
MY.NET.206.102:1995 -> 200.60.137.73:27374
04/07-06:17:01.360715  [**] Possible trojan server activity [**]
200.60.137.73:27374 -> MY.NET.206.102:1995
04/07-07:12:41.741569  [**] Possible trojan server activity [**]
MY.NET.233.146:1382 -> 80.60.97.219:27374
04/07-07:47:14.836802  [**] Possible trojan server activity [**]
MY.NET.100.165:80 -> 193.134.254.145:27374
```

```
04/07-08:20:24.312231  [**] Possible trojan server activity [**]
62.131.15.233:27374 -> MY.NET.233.146:1382
04/07-09:31:01.518093  [**] Possible trojan server activity [**]
4.19.237.4:27374 -> MY.NET.24.34:80
04/07-09:31:01.518245  [**] Possible trojan server activity [**]
MY.NET.24.34:80 -> 4.19.237.4:27374
```

By looking at the above samples it seems as if this alert is triggered as soon as a destination or source port of 27374 is used, which is well known to be used by Sub-7 trojan. This signature unfortunately also generates quite a lot of false alerts as this port can be used in “normal” connections as it can be seen in the last sample above where the traffic looks to be a normal HTTP connection.

Conclusion: It seems as if we’re seeing a lot of scans coming in and going out to port 27374 – probably just a probing for Sub-7. Although some host seem more suspicious than others: MY.NET.105.48 is receiving a lot of connections from different source ports to port 27374 which could be an indication for Sub-7 connections. Another suspicious host seems MY.NET.206.102 which is talking from time to time with two external IP on these ports. MY.NET.233.146 should also be further investigated as he’s talking two multiple external hosts on these ports.

Alert: Back Orifice

Reported 3 times, severity high

Analysis: Back Orifice is a Trojan which infects Windows machines and can be seen in the wild quite often. Unfortunately, the Snort signature which was used here is not known. If the signature is very specific, these alerts should be treated seriously. Two IPs seem to be infected, MY.NET.168.70 (attacked by 61.74.67.46) and MY.NET.162.25 (attacked by 63.250.207.55). At least there are connections to port 31337 (which is known to listen for Back Orifice connections). Checking the alerts reveals that MY.NET.168.70 only triggered this one alert, MY.NET.162.25 although triggered more than 40 alerts. This host will be investigated further in the chapter ‘Interesting Hosts’

Conclusion: It would be wise to check these two machines with an up-to-date anti virus scanner. Establishing and enforcing an anti virus policy should also be considered.

Alert: site exec - Possible wu-ftpd exploit - GIAC000623

Reported once, severity low

Analysis: This alert was only reported once. It was an incoming connection from 24.47.223.9:3353 to MY.NET.225.78:21. Andy Johnston is commenting on this particular signature, which is triggered as soon as an incoming connection to port 21 contains the “site exec” content. With the site exec command it is possible to execute commands on the FTP server and therefore create or open up some vulnerability.

Conclusion: The attacked IP here is only listed twice in the alert log, once for the possible wu-ftpd exploit and the other time for an SMB Name Wildcard alert. According to this information and clues that this IP could even be used as an FTP, the severity can be regarded as very low. This most probably is a false positive.

Alert: SNMP public access

Reported 148 times, severity medium

Analysis: The Simple Network Management Protocol (SNMP) is implemented and used for network device management. Most network devices (routers, switches, etc.) are manageable by SNMP, which means you can get status information of these devices as well as configure them directly via SNMP. The access control to these settings is managed by a community string which is often set to 'public' and 'private' as a factory default (and which doesn't get changed a lot of times). The community string can be regarded as a kind of password which is sent unfortunately unencrypted. The default community string should always be changed to make it harder for an attacker, virus or worm to get access to this management possibility.

Conclusion: SNMP access should be restricted; especially from the Internet (the firewall should drop these packets). The community string should be handled like a password and therefore needs to be changed from time to time. Administrators should use SNMP wisely and always be aware of its "insecurity".

Alert: IRC evil - running XDCC

Reported 98 times, severity medium

Analysis: 17 different sources within the MY.NET network are listed for potentially running XDCC. XDCC is used for a kind of file transfer on IRC: *"An XDCC server is a server that allows others to access files on your computer that you have organized into packs."* XDCC is also often used by hacker tools, esp. distributed denial of service, to send commands from the master to the bots.
It's very likely, that XDCC is used in the "normal" way, therefore for sharing files. Warez is often distributed in this way and very popular with students.

Conclusion: The IP's from this list should be added to a watch list. If other alerts are coming up for one of these IP's, especially such alerts which are connected to Trojans and distributed denial of service tools, the IP should be further investigated.

Top Talkers

The top talkers were selected according to the number of alerts they generated. It is assumed that the more alert a system generates, the more traffic was flowing to and from this system.

IP	Source	Destination	Total alerts
62.13.46.133	0	49396	49396
217.75.98.40	0	42195	42195
216.152.64.213	0	27536	27536
216.152.64.143	0	26837	26837
216.152.64.158	0	24214	24214
MY.NET.100.165	60	20596	20656
MY.NET.201.58	8495	7955	16450
194.23.46.226	0	10695	10695
MY.NET.201.218	0	9866	9866

212.179.85.47	9633	0	9633
66.42.68.210	4664	4588	9252
216.39.48.2	8909	0	8909
212.179.88.179	5972	0	5972
MY.NET.196.51	0	5969	5969
MY.NET.240.78	5960	4	5964
156.17.180.132	4493	0	4493
MY.NET.225.174	0	4456	4456
129.44.184.64	4444	0	4444

Our number one top talker, IP 62.13.46.133 is also added to the ‘Interesting Hosts’ list and examined further in the ‘Interesting Hosts’ chapter.

Interesting Out of Spec Traffic

The Out-of-Spec logs files contain packets which were generated for one of the following three reasons:

- Corrupted packet
- Crafted packets
- Implementation of the Explicit Congestion Notification standard (RFC2481)

When looking through the OOS log files I noticed that most OOS log entries correspond to traffic entering our network. Only a small number of packets are listed which are destined at an IP outside our network and nearly all of them are coming from MY.NET.12.2 and MY.NET.12.4

```
04/09-22:35:44.095147 MY.NET.12.2:25 -> 172.128.74.57:1328
TCP TTL:255 TOS:0x0 ID:1529 IpLen:20 DgmLen:40
12***R** Seq: 0x4B4BF113  Ack: 0x0  Win: 0x0  TcpLen: 20

04/10-01:49:08.535790 MY.NET.12.2:25 -> 202.84.36.63:1612
TCP TTL:255 TOS:0x0 ID:21389 IpLen:20 DgmLen:40
12***R** Seq: 0xA8021065  Ack: 0x0  Win: 0x0  TcpLen: 20

04/10-22:09:38.002614 MY.NET.12.4:993 -> 68.55.193.137:29277
TCP TTL:255 TOS:0x0 ID:28954 IpLen:20 DgmLen:40
12***R** Seq: 0xEFA2EC19  Ack: 0x0  Win: 0x0  TcpLen: 20

04/10-23:34:47.253730 MY.NET.12.4:143 -> 68.55.244.19:64817
TCP TTL:255 TOS:0x0 ID:14048 IpLen:20 DgmLen:40
12***R** Seq: 0x58EA6608  Ack: 0x0  Win: 0x0  TcpLen: 20
```

On all these log entries, a reset is sent to the other communication partner. This is nothing abnormal, but here, the reserved TCP flag bits (bit 1 and 2) are set. These bits are newly used by the ECN standard. Not many vendors have included these flags in their network devices so they can still be considered out of the spec. Also see Toby Millers paper about ECN and the impact on intrusion detection.

We also can see crafted packets, like this NULL-scan

```
04/09-00:11:51.209137 66.36.129.232:64712 -> MY.NET.224.202:3554
TCP TTL:111 TOS:0x0 ID:42997 IpLen:20 DgmLen:40 DF
******** Seq: 0xE6BCD4C8  Ack: 0xEE4D40EF  Win: 0x0  TcpLen: 0
```

or packets where strange combinations of TCP flags are used as it can be seen in the following packet where the SYN, ACK, Urgent, FIN and RST flags are set.

```
04/09-13:15:00.634221 212.62.81.102:2137 -> MY.NET.201.106:4662
TCP TTL:113 TOS:0x0 ID:34830 IpLen:20 DgmLen:40 DF
1*UA*RSF Seq: 0x6A4D4095  Ack: 0x36173617  Win: 0x5010  TcpLen: 52
UrgPtr: 0x7110
TCP Options (1) => EOL
```

It’s interesting to see that a lot of traffic which is listed in these log files seems to be connected to peer-to-peer software like KaZaA or BearShare. The following packet shows some invalid traffic as both source and destination port are set to 0, which is reserved and should not be used.

```
04/09-23:17:17.950347 168.122.195.80:0 -> MY.NET.233.146:0
TCP TTL:119 TOS:0x0 ID:26940 IpLen:20 DgmLen:439 DF
******** Seq: 0x2010000  Ack: 0x0  Win: 0x0  TcpLen: 0
47 45 54 20 2F 2E 68 61 73 68 3D 30 63 30 30 35  GET /.hash=0c005
37 64 66 37 30 36 66 61 64 36 62 32 38 38 61 64  7df706fad6b288ad
37 31 38 36 66 31 37 65 36 36 30 32 34 31 63 37  7186f17e660241c7
34 38 63 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F  48c HTTP/1.1..Ho
73 74 3A 20 31 33 30 2E 38 35 2E 32 33 33 2E 31  st: MY.NET.233.1
34 36 3A 31 33 38 32 0D 0A 55 73 65 72 41 67 65  46:1382..UserAge
6E 74 3A 20 4B 61 7A 61 61 43 6C 69 65 6E 74 20  nt: KazaaClient 
4E 6F 76 20 20 33 20 32 30 30 32 20 32 30 3A 32  Nov  3 2002 20:2
39 3A 30 33 0D 0A 58 2D 4B 61 7A 61 61 2D 55 73  9:03..X-Kazaa-Us
65 72 6E 61 6D 65 3A 20 6D 65 0D 0A 58 2D 4B 61  ername: me..X-Ka
7A 61 61 2D 4E 65 74 77 6F 72 6B 3A 20 4B 61 5A  zaa-Network: KaZ
61 41 0D 0A 58 2D 4B 61 7A 61 61 2D 49 50 3A 20  aA..X-Kazaa-IP: 
31 36 38 2E 31 32 32 2E 31 39 35 2E 38 30 3A 32  168.122.195.80:2
39 30 32 0D 0A 58 2D 4B 61 7A 61 61 2D 53 75 70  902..X-Kazaa-Sup
65 72 6E 6F 64 65 49 50 3A 20 31 36 38 2E 31 32  ernodeIP: 168.12
32 2E 32 32 37 2E 32 34 32 3A 32 37 33 34 0D 0A  2.227.242:2734..
52 61 6E 67 65 3A 20 62 79 74 65 73 3D 34 37 36  Range: bytes=476
35 30 38 32 33 37 2D 34 37 37 35 35 36 35 31 37  508237-477556517
0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 63 6C  ..Connection: cl
6F 73 65 0D 0A 58 2D 4B 61 7A 61 61 2D 58 66 65  ose..X-Kazaa-Xfe
72 49 64 3A 20 32 38 32 30 31 32 39 0D 0A 58 2D  rId: 2820129..X-
4B 61 7A 61 61 2D 58 66 65 72 55 69 64 3A 20 33  Kazaa-XferUid: 3
49 71 6A 55 6F 54 51 63 36 57 42 72 74 38 6C 68  IqjUoTQc6WBrt8lh
6E 78 6A 59 48 63 4D 68 53 2B 54 45 73 70 49 67  nxjYHcMhS+TEspIg
56 64 71 2B 33 65 2F 63 72 49 3D 0D 0A 0D 0A     Vdq+3e/crI=....
```

The out of spec log files can provide valuable information, especially about crafted packets. The logs here show that the border firewall could be further tuned to drop invalid packets (i.e. like packets targeted at port 0, having invalid TCP flags, etc.) as they are mostly used for reconnaissance. If a packet gets corrupted on its way to the target, there is also no problem in dropping the packet as it will be dropped sooner or later – there is not much sense in sending these packets further on.

Interesting Hosts

This chapter lists some systems which have been added to the “Interesting Hosts” list during earlier analysis.

MY.NET.100.165

MY.NET.100.165 seems to be an important or suspicious machine – more than 20'000 alerts were triggered by this system. According to an alert, this seems to be a web server called 'CS WEBSERVER'. An analysis of this IP and its corresponding alerts is shown in the following table

Alert	Count
CS WEBSERVER - external web traffic	19794
CS WEBSERVER - external ftp traffic	338
Watchlist 000222 NET-NCFC	276
SMB Name Wildcard	95
Possible trojan server activity	75
Watchlist 000220 IL-ISDNNET-990517	47
spp_http_decode: IIS Unicode attack detected	24
Queso fingerprint	7
NMAP TCP ping!	4
IDS552/web-iis_IIS ISAPI Overflow ida nosize	1

Looking at all these alerts seems as if it could be really possible that this machine is a web server. Especially the top two alerts strengthen this assumption. The "possible trojan server activity" can be ignored as all these alerts are most probable false positives. As mentioned in an earlier chapter, this alert is triggered as soon as port 27374 is used as source or destination port. All the alerts here have either port 80 as source and 27374 as destination or vice versa which indicates that we are dealing with normal HTTP traffic.

MY.NET.162.25

MY.NET.162.25 was added to the "Interesting Hosts" list because a "Back Orifice" alert was detected as well as 40 other alerts in conjunction with this host. Let's see what kind of alerts these were:

Alert	Count
SMB Name Wildcard	22
Possible trojan server activity	6
External RPC call	3
DDOS shaft synflood incoming	2
Back Orifice	1

The "Possible trojan server activity" is suspicious here. It looks as if multiple connections were established between MY.NET.162.25 and an external IP (62.175.146.2). The connections are always coming from port 27374 and going to port 95 (or vice versa). All six alerts were generated within 30 minutes.

Port 95 is associated with a service called 'SUPDUP', an old UNIX subsystem, which was an extension to Telnet for out-of-band graphics control items. Cisco also uses this port for their own purposes. Doing some research on this IP shows that it belongs to a telephone company in Spain.

```
% 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:      62.175.144.0 - 62.175.159.255
netname:      RETENET
descr:        AUNA S.A.U,
descr:        Avenida Diagonal, 579
descr:        Barcelona 08014
descr:        Spain
country:      ES
admin-c:      TA718-RIPE
tech-c:       TA718-RIPE
status:       ASSIGNED PA
mnt-by:       AUNA-MNT
mnt-lower:    AUNA-MNT
remarks:      --------------------------------------------------
remarks:      for peering questions:   techauna@auna.es
remarks:      for net abuse questions: abuse@auna.es
remarks:      --------------------------------------------------
changed:      techauna@auna.es 20020318
source:       RIPE

route:        62.174.0.0/15
descr:        Retevision SA
origin:       AS8761
notify:       techauna@auna.es
mnt-by:       AUNA-MNT
changed:      techauna@auna.es 20010615
source:       RIPE

role:         Techauna AUNA
address:      Avenida Diagonal, 579
address:      Barcelona 08014
address:      Spain
phone:        +34 93 502 0000
fax-no:       +34 93 502 2809
e-mail:       techauna@auna.es
admin-c:      EES12-RIPE
tech-c:       AGS30-RIPE
tech-c:       EES12-RIPE
tech-c:       ABRP1-RIPE
nic-hdl:      TA718-RIPE
notify:       techauna@auna.es
mnt-by:       AUNA-MNT
remarks:      --------------------------------------------------
remarks:      for net abuse questions please contact:
remarks:      abuse@auna.es
remarks:      --------------------------------------------------
changed:      techauna@auna.es 20030312
source:       RIPE
```

They also offer ISP services to their customers which make me believe that we are dealing with some sort of home or business user. A reverse lookup unfortunately doesn't reveal much more as the name '2-146-IBA.red.retevision.es'.

The "External RPC call" is triggered because we have an incoming connection coming to port 111. It's hard to say if this attack (or scan) was successful, but most probably it wasn't. To make sure, incoming RPC calls should be dropped at the border (unless really needed).

This host should be scanned for Trojans and checked because we could be dealing with a possible trojan attack. All other events triggered in conjunction with MY.NET.162.25 seem to be false positives.

MY.NET.105.48

MY.NET.105.48 generated quite a number of alerts.

Alert	Count
Watchlist 000222 NET-NCFC	115
TFTP - External TCP connection to internal tftp server	101
High port 65535 tcp - possible Red Worm – traffic	95
SUNRPC highport access!	77
DDOS mstream client to handler	74
External RPC call	69
connect to 515 from outside	69
Possible trojan server activity	61
Watchlist 000220 IL-ISDNNET-990517	54
SMB Name Wildcard	10
IRC evil - running XDCC	1

The system should be checked if a TFTP server is running. If so, packets destined at this port and machine should be dropped at the firewall (if this service should not be available to the public).

The "High port 65535 tcp – possible Red Worm traffic" is prone to false positives as it triggers on activity on port 65535. Investigating these alerts here further makes me believe that we also having a lot of false positives. No traffic can be seen where reflexive ports have been used which would require further investigations.

We have no IP address which is attacking or is attacked by this system often. Most attacks are unique in source IP and destination IP. This makes me believe that we are dealing mainly with noise generated by not very specific IDS rules.

Never the less, the system should be checked with an anti virus software to make sure, no Trojans are installed on this system.

MY.NET.233.146

Let's investigate a little further for a possible "problem child".

Alert	Count
Watchlist 000220 IL-ISDNNET-990517	92
SMB Name Wildcard	53
Possible trojan server activity	5
High port 65535 tcp - possible Red Worm – traffic	2

As mentioned earlier, the "High port 65535 tcp – possible Red Worm – traffic" is probe to false positives. But in depths look shows that we are most likely dealing with an infected host. 168.122.195.80 is talking to this post, both parties using port 65535. A further investigation of the scan files shows that these alerts were generated by a xmas scan originated from port 65535 targeted at port 65535.

The “Possible trojan server activity” alerts shows that all alerts triggered on traffic with the external IP 66.20.83.108 which seems to be an ADSL connection from Bell South. The traffic was always coming from MY.NET.233.146:1382 targeted at port 27374 of the external user, which could indicate that we’re seeing normal peer-to-peer traffic.
Never the less, the system should be watched.

80.14.81.238

This IP was added to the ‘Interesting Hosts’ list because he scanned systems within our network for more than 8000 times. Querying the database shows that this host never triggered an alert.

```
% 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:      80.14.81.0 - 80.14.81.255
netname:      IP2000-ADSL-BAS
descr:        BSFNY111 Fontenay Bloc1
country:      FR
admin-c:      WITR1-RIPE
tech-c:       WITR1-RIPE
status:       ASSIGNED PA
remarks:      for hacking, spamming or security problems send mail to
remarks:      postmaster@wanadoo.fr AND abuse@wanadoo.fr
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20020219
changed:      gestionip.ft@francetelecom.com 20030318
source:       RIPE

route:        80.14.0.0/16
descr:        France Telecom
descr:        Wanadoo Interactive
remarks:      --------------------------------------------
remarks:      For Hacking, Spamming or Security problems
remarks:      send mail to      abuse@francetelecom.net
remarks:      --------------------------------------------
origin:       AS3215
mnt-by:       RAIN-TRANSPAC
mnt-by:       FT-BRX
changed:      karim@rain.fr 20011221
source:       RIPE

role:         Wanadoo Interactive Technical Role
address:      WANADOO INTERACTIVE
address:      48 rue Camille Desmoulins
address:      92791 ISSY LES MOULINEAUX CEDEX 9
address:      FR
phone:        +33 1 58 88 50 00
e-mail:       abuse@wanadoo.fr
e-mail:       technical.contact@wanadoo.com
admin-c:      WITR1-RIPE
tech-c:       WITR1-RIPE
nic-hdl:      WITR1-RIPE
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20010504
changed:      gestionip.ft@francetelecom.com 20010912
changed:      gestionip.ft@francetelecom.com 20011204
```

```
changed:      gestionip.ft@francetelecom.com 20030428
source:       RIPE
```

The logs show, that this attacker SYN scanned a huge number of IP addresses within our network. Hew as looking for activities on port 135 (99%), port 139 (0.5%) and port 445 (0.5%). These ports are most probably scanned to find vulnerable Windows systems (on October 18, 2002 a DoS for RPC service on Windows 2000 SP3 was detected and was probably scanned for here).

MY.NET.70.136

This IP was added to the 'Interesting Hosts' list because he scanned systems within our network for more than 5500 times.

Alert	Count
SMB Name Wildcard	2

This system should most probably be watched closer. Two SYN and more than 5000 UDP scans were launched against external systems at multiple ports including port 17200 (40%), port 12823 (10%), port 45379 (10%), 62310 (10%) and many more.

The OOS log files list one occurrence with this host on 04/08:

```
04/08-13:33:01.628364 172.163.112.236:1389 -> MY.NET.70.136:5156
TCP TTL:50 TOS:0x0 ID:34810 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x93E9D963  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 745113 0 NOP WS: 0
```

This packets shows that the two reserved TCP flags are set. Probably there are routers on it's travel route through the Internet which supported the new Explicit Congestion Notification (ECN). This packet shouldn't worry any longer.

Never the less, it is interesting to see so many scans originating from one host. Probably this system is used for some kind of load balancing where these scans are used as probes. But it is also possible that there is some malicious activity behind all this. What's most interesting is the fact that only 19 different IP were scanned, but they were scanned regularly (most of them about the same number of times). The following table lists the 10 most scanned hosts.

IP	Reverse Lookup	Scans
80.139.178.30	p508BB21E.dip.t-dialin.net	498
80.200.118.241	241.118-200-80.adsl.skynet.be	498
4.63.63.210	lsanca1-ar23-4-63-063-210.lsanca1.dsl-verizon.net	488
217.229.198.234	pD9E5C6EA.dip.t-dialin.net	486
80.126.148.112	a80-126-148-112.adsl.xs4all.nl	486
213.84.36.25	a213-84-36-25.adsl.xs4all.nl	484
129.44.44.43	pool-129-44-44-43.ny325.east.verizon.net	479
68.12.68.108	ip68-12-68-108.ok.ok.cox.net	398
4.18.167.2	Not found	369

199.203.245.55	DSL199-203-245-55.bb.netvision.net.il	348

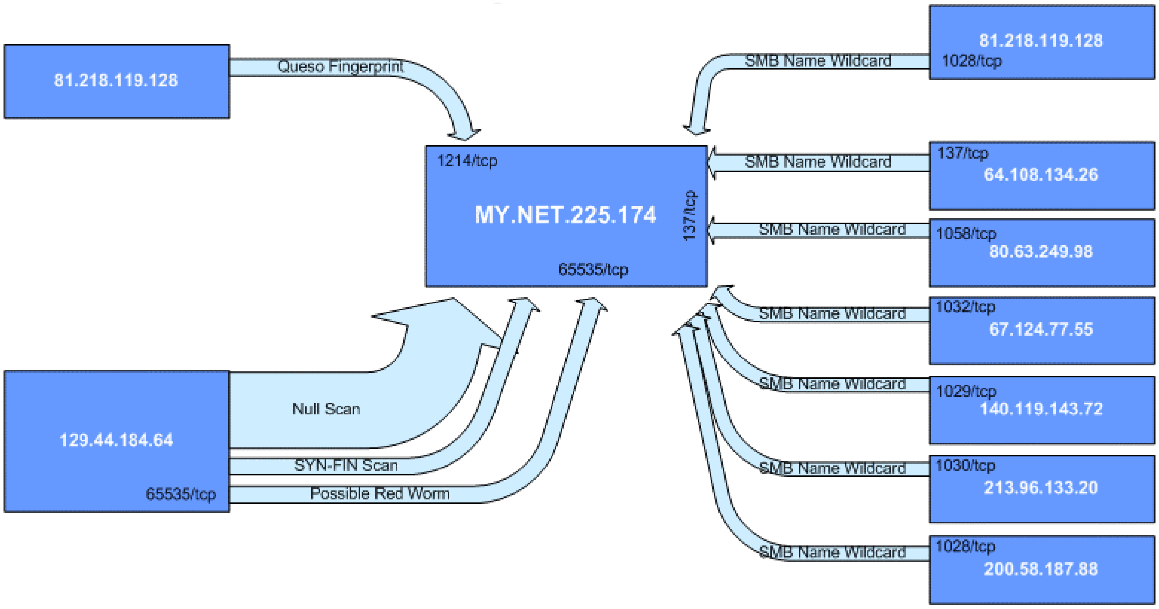
All of the scanned IP's looks as if they are belonging to some ISP's offering ADSL or dialin connections. As I said earlier, this system should be monitored closely and probably also scanned with an anti-virus software.

MY.NET.225.174

This IP was added to the 'Interesting Hosts' list because he was scanned for more than 5000 times.

Alert	Count
High port 65535 tcp - possible Red Worm - traffic	17
Probable NMAP fingerprint attempt	13
SMB Name Wildcard	12
Queso fingerprint	5

This IP is also one of the probably infected Red Worm systems. We have a number of connections involving connections from and to port 65535. The system should be scanned and added to a watch list. The linkgraph shows the different involved systems and their access to MY.NET.225.174. The host 129.44.184.64 definitively looks very suspicious here. A fast look at the logs shows that he only "attacked" MY.NET.225.174 and no other systems within our network. Reverse lookup unfortunately can't resolve the IP, whois tells us that the IP belongs to a company called "Verizon Global Networks, Inc.", an ISP provider.



62.13.46.133

This host was added to the ‘Interesting Hosts’ list because he’s listed as the number one on the “Top Talkers” list. He holds the number one position there for having triggered the most number of alerts as a target.

Alert	Count
TCP SRC and DST outside network	49416

This system triggered one alert for nearly 50’000 times. As the alert says, target and source IP address are not within our network. We are definitively dealing with spoofing or misconfiguration. The alert was never triggered with 62.13.46.133 acting as source. It therefore looks as if we are attacking this IP from our network with spoofed IP’s.

A whois lookup tells us, that this IP belongs to a company in Sweden called “Gate Company SA”.

```
% 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:      62.13.46.0 - 62.13.46.255
netname:      GATECOMPANY-SE-NET
descr:        Gate Company AB
descr:        Stockholm
country:      SE
admin-c:      ME4036-RIPE
tech-c:       ME4036-RIPE
status:       ASSIGNED PA
mnt-by:       AS8434-MNT
changed:      kandra@utfors.se 20010724
source:       RIPE
```

A reverse lookup shows that the IP is translated into die.nu – Looking up their webpage shows some most interesting things. Die.nu is known as shells.se, a service which offers shell accounts, e-mail addresses and hosting services. This would definitively make sense as a target for a denial of service attack which is most often used with spoofed IP addresses. By looking at the timestamps of the alerts shows, that all 49416 alerts were generated within 2 minutes, which means we had an average of 411 alerts each second.
Unfortunately the scan logs don’t hold more information about a scan for the network 62.13.46.0/24.
It seems important to adjust the firewall rules to drop outbound traffic which is spoofed to eliminate such attacks to the outside world.

Conclusion and Recommendations

After all the thousands of analyzed log entries, I believe the University has a bad perimeter defense in place. At the same time, the IDS signatures have to be further tuned and adjusted to be more specific on attacks. Signatures which only trigger on certain involved port numbers are prone to a very high rate of false positives.

First, I recommend establishing a security policy. What kinds of services are needed to be accessible from the outside? What kind of traffic is allowed and what are the restrictions for the students? What about anti virus solutions? It would be a good idea to enforce the use of an anti-virus solution to reduce the risk of malware infections.

Second, I recommend improving perimeter security, therefore adjusting the firewall rules. If a firewall is in place at the moment, it definitively looks like Swiss cheese. The following improvements should be considered:

- Drop invalid packets at the border router
- Establish a content filter to drop known worms (and possibly also peer-to-peer traffic if the security policy does not allow file sharing via p2p software)
- Set up firewall rules according to the security policy. Therefore, if there is no need to be able to access Windows shares from an external system; incoming traffic to these well known ports (135-139) should be dropped.

Finally, the users at the University should be educated about the risk and danger of using the Internet, about open Windows shares, p2p software, malware and the legal issues of sharing music, application and other copyright protected information.

At the same time, a network assessment should be considered to define which systems need to be protected and how the Intrusion Detection system can be tuned to help detecting critical attacks. A process for checking the log files and acting in case of an intrusion should be established – An IDS system alone doesn't help to improve the level of IT security if nobody is watching and maintaining the system.

Methodology and Tools

For easier analysis the log files were scanned, splitted and inserted into a relational database. With this setup, in-depth analysis and especially statistical analysis is very easy and can be done quite fast. With a Snort installation it is even easier as Snort can directly log to a database – Unfortunately, the log files here were only available as Snort text log files which had to be parsed first.
A MySQL 4.1 database was chosen for its simplicity, very fast operation and availability. Version 4.1 has the advantage of supporting sub-queries which made some analysis more easy and comfortable. MySQL version 4.1 is still considered alpha, but we never experienced any problems.
The database design which was chosen is simple and straight forward and inspired by the setup Snort itself uses.

After the successful load of the scans and alerts log files into the database, a first overview was established by querying the database for the alert type which were stored. With the help of the database, getting an overview of the different types of alerts as well as their number of occurrences, the top alerts were picked and analyzed further. During this analysis, different suspicious host were found and added to the "Interesting Hosts" list for later investigations. After a successful analysis of the alerts which were recorded the most often, the most suspicious alerts were analyzed more closely to look for compromised hosts. The scan logs were also used to find hosts which generated a lot of traffic. The information in the "Out of Spec" logs helped to determine if a certain alert or scan should be considered a false positive, i.e. was triggered due to a network error, ECN or if we had a crafted packet.

The “Interesting Hosts” list was analyzed on a host-basis (in contrary to the earlier analysis which was alert-driven). Some hosts could be classified for a potential compromise and others needed investigation on the machine itself for malware or Trojans.
The methodology which was used seemed to fit and worked very well during this assignment. It would have been great to have more log details, like a tcpdump session of all the suspicious traffic. This would have enabled an even more in-depth look.

Technical Implementation

The log files were parsed with PHP and inserted into MySQL. PHP was chosen because a lot of experience was available and some scripts for internal IDS projects were already developed in a similar form.

The PHP code for loading the alert log files is quite simple. For easier database handling, the PHP Base Library was used which provides a very useful database class. To load the scan log files, a similar script was used.

```
<?
    include("include/prepend.php3");

  $db = new main_db;

  $logdir = "/www/gcia/logs/";
  $filename[1] = "alert.030407";
  $filename[2] = "alert.030408";
  $filename[3] = "alert.030409";
  $filename[4] = "alert.030410";
  $filename[5] = "alert.030411";


  for ($i=1; $i<=count($filename); $i++) {

  $handle = fopen($logdir.$filename[$i],"r");
  while (!feof($handle)) {
         $buffer = fgets($handle, 4096);
         echo($buffer."<br>");
         list($date, $alert, $details) = split("\[\*\*\]",$buffer);

         if (strpos($details,":")>1) {
                list($src_ip, $src_port, $dst_ip, $dst_port) =
                split("\:|\-\>",$details);
         }
         else {
                list($src_ip,$dst_ip) = split("\-\>",$details);
                $src_port = 0;
                $dst_port = 0;
         }

         // check if signature exists
         $db->query("select sig_id from signatures
                       where name='$alert'");
         if ($db->num_rows()<1) {
                $db->query("insert into signatures (name,count)
                              values ('$alert',1)");
                $db->query("select sig_id from signatures
                              where name='$alert'");
         }
```

```
        $db->next_record();
        $sig_id = $db->f("sig_id");
        $db->query("update signatures set count=count+1
                    where sig_id=".$sig_id);

        // generate alert
        list($month,$day,$hour,$minute,$sec,$micro_sec) =
               split("\/|\-|\:|\.",$date);
        $timestamp = "2003".$month.$day.$hour.$minute.$sec;
        $db->query("insert into alerts (generated, ts, sig_id) values
                    ('$timestamp','$date',$sig_id)");
        $db->query("select alert_id from alerts where sig_id=$sig_id
                           and generated='$timestamp'");
        $db->next_record();
        $alert_id = $db->f("alert_id");

        // generate details
        if ((int)$dst_port<1) $dst_port = 0;
        if ((int)$src_port<1) $src_port = 0;
        $db->query("insert into details
               (alert_id,src_ip,src_port,dst_ip,dst_port)
               values
               ($alert_id,'$src_ip',$src_port,'$dst_ip',$dst_port)");
  }
  fclose($handle);
  }
?>
```

References

Anonymous. “Distributed Denial of Service (DDoS) Attacks/tools”. April 14, 2003
URL: http://staff.washington.edu/dittrich/misc/ddos/ (April 28, 2003)

Anonymous. “MySQL – The world’s most popular Open Source Database”
URL: http://www.mysql.com (April 24 2003)

Anonymous. "Neohapsis Ports List."
URL: http://www.neohapsis.com/neolabs/neo-ports/neo-ports.html (April 28, 2003)

Anonymous. “PHP: Hypertext Preprocessor”
URL: http://www.php.net/ (April 25 2003)

Anonymous. “PHP Base Library”
URL: http://phplib.sourceforge.net/index.php3 (April 25 2003)

Anonymous. "Trojan and Remote Access Service Ports.". Oct. 20 2002.
URL: http://doshelp.com/trojanports.htm (April 21, 2003).

Anonymous. “XDCC”. March 28, 2003
URL: http://members.aol.com/lamesn/myhomepage/XDCC.htm (March 28, 2003)

Alexander, Bryce. “Intrusion Detection FAQ – Port 137”. May 10, 2000
URL: http://www.sans.org/resources/idfaq/port_137.php (April 21 2003)

Baumann, Reto. “Scan Methods”. April 30, 2003
URL: http://security.rbaumann.net/scans.php?sel=1 (March 30, 2003)

Beardsley, Tod A. “GIAC Practical”. May 8, 2002
URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc (April 19 2003)

CERT. “CERT Incident Note IN-2000-02”. March 3, 2000
URL: http://www.cert.org/incident_notes/IN-2000-02.html (April 25 2003)

CERT. “CERT® Incident Note IN-2000-05”. May 2, 2000
URL: http://www.cert.org/incident_notes/IN-2000-05.html (April 28, 2003)

CERT. “CERT® Advisory CA-1999-17 Denial-of-Service Tools”. December 28, 1999
URL: http://www.cert.org/advisories/CA-1999-17.html (April 28, 2003)

CERT. “SNMP Vulnerabilities – FAQ”. February 13, 2002
URL: http://www.cert.org/tech_tips/snmp_faq.html (March 28, 2003)

Dell, J Anthony. "Adore Worm – Another Mutation." Apr. 6 2001.
URL: http://rr.sans.org/threats/mutation.php (April 27, 2003).

Cisco. "How to Block the ‘Code Red’ Worm." Cisco Tech Notes. Sep. 5 2002
URL: http://www.cisco.com/warp/public/63/nbar_acl_codered.shtml#1 (April 28, 2003)

IANA. "Port Numbers." Assigned Numbers. April 27, 2003
URL: http://www.iana.org/assignments/port-numbers (April 27, 2003).

Johnston, Andy (and others). “Global Incident Analysis Center”. June 20,2000
URL: http://www.sans.org/y2k/063000.htm (April 28, 2003)

Miller, Toby. “ECN and it’s impact on Intrusion Detection”, Nov 3, 2000
URL: http://www.securityfocus.com/infocus/1205 (May 2, 2003)

Russel, Ryan: “DDOS shaft synflood”. Dec 28, 2001
URL: http://www.mcabee.org/lists/snort-users/Dec-01/msg00639.html (April 28, 2003)

SamSpace.org. “Usefull Tools”. April 28, 2003
URL: http://www.samspade.org/t/ (April 28, 2003)

Symantec. “DDoS.Mstream”. April 28, 2003
URL: http://securityresponse.symantec.com/avcenter/venc/data/ddos.mstream.html (April 28, 2003)

Symantec. “Information on Back Orifice and NetBus”, April 28, 2003
URL: http://www.symantec.com/avcenter/warn/backorifice.html (April 28, 2003)