Global Information Assurance Certification Paper

GIAC Certified Intrusion Analyst (GCIA)
Practical Assignment
Version 4.1

Listening to White Noise

Looking for patterns of malicious traffic

Jorge D. Ortiz-Fuentes

November 16, 2004

Abstract

This document contains the full detailed analysis of the IDS logs captured at the University between April 20, 2004 and April 22, 2004.

The paper is divided in three parts. The first part is an executive summary of the analysis. The second part describes the network, analyzes the IDS logs and perform an in-depth analysis of three detects. Finally, part three covers the methodology used for getting the results.

Contents

Acknowledgments

The unquestionable help that Rosa and Lidia have provided me with, through their patience and unconditional support, has been the decisive factor for writing this paper.

I also thank David and Raul. Working together as a group is making us all improve and extend our knowledge about security.

Finally, I sincerely thank all the people that have put their knowledge and work in developing the free tools that I use every day and that are both an important part of what I explain in this paper and the applications used for writing it.

Part I
Executive Summary (10 points)

This document contains the full detailed analysis of the IDS logs captured at the University between April 20, 2004 and April 22, 2004.

The systems that offer the main network services to the alumni, faculty and staff have not been involved in any major security events. However, malicious activity has been detected affecting to several systems. Many systems could be running programs that provide unauthorized access to them, while some others could be running tools used to perform distributed attacks against Internet sites.

The Chief Security Officer should initiate actions so the affected systems are investigated and recovered to a *clean* state. Other events should be investigated as well as recommended through this document.

Additionally, in order to increase the security of the University network and information systems, I would recommend doing the following things:

- Increase security awareness among the alumni, faculty, and staff.
- Train and dedicate people to review the alerts generated by the intrusion detection systems.
- Create a computer emergency response team that is responsible for investigating suspicious activity.
- If possible for this University install a perimeter firewall. If it is already installed configure it properly: deny by default.
- Configure systems and network elements so other events are captured. This will help in case of an eventual investigation.
- Keep a full network capture for as long as it is possible.
- Be proactive in patching the systems.
- Stablish or review the system configuration policy. Require personal firewalls, antivirus, and safe defaults.

Part II
Detailed Analysis (70 points)

1 Data to Analyze

The log files analyzed in this documment were generated during three days starting april 20th, 2004 and finishing april 22nd, 2004. The nine files used can be

downloaded from http://isc.sans.org/logs and their names and MD5 hashes are:

Name	MD5
alert.040420.gz	d6d0fd09056958d73d26090f913a0633
alert.040421.gz	5da640f15825446400aff861aa4a8158
alert.040422.gz	bbf45a73fa3bea9b58615e304fb6b75e
oos_report_040420.gz	b927c85d715ac534200a4060acd90b24
oos_report_040421.gz	2b90f5b7fc20e1cfb9da7dfea39bc513
oos_report_040422.gz	a68ac2316e5df4f8b82468a09e580786
scans.040420.gz	8595f1be4f4f8e352bd0dfcd9fbf5770
scans.040421.gz	5580a3066e21d36a38ef43d914f5e411
scans.040422.gz	17b0535b52b1606cc316c1cf8865bb49

These data have been generated by a snort instance. However, there is no information about the version of either Snort or the ruleset.

2 Topology

The first task I had to acomplish was extracting information about the topology for a better understanding of the logs. It would be very valuable to obtain a picture about the physical topology from the layer two addresses (MAC), however this information was not available in any of the files used for this analysis. It is possible from the information present in the logs to establish the logical topology of the machines involved in them, though. In order to do so, I extracted the IP address information from the three different file types: alerts, oos, and scans. Each file type had a different format and there were two different types of entries in the alert files —one for the portscans and another for the rest of the alerts. I wrote the Perl script that can be found in section 9.1. This script uses Perl's capability to work with regular expressions and extract data using them.

I noticed that in the alert and oos files, the first two bytes of the IP address space of the University begun with `MY.NET`, but this was not the case with the scan files that contained fully specified IP addresses. It seems that every entry of the scan files, has an IP address with the form `130.85.x.y` as can be extracted from the output of the following commands:

```
# wc -l scans.040420
2042803 scans.040420
# grep -v '130\.85' scans.040420 | wc -l
     21
```

Futher review of the 21 lines that do not contain that type of address confirms that they are incomplete or bad formed entries. This allows me to assume that `MY.NET` is the same as `130.85`, which could be a class B network[1]. The script replaces those values in order to have an homogeneus set of data.

[1]Although the network 130.80.0.0/24 belongs to the cannonical class B address space, the data might have been sanitized so I cannot trust this hint.

```
# topology.pl alert.04042* oos* scans.04042*
2098814 different IP addresses.
15747 internal IP addresses.
2083067 external IP addresses.

------------
Top 5 mail:
------------
IP address      Frequency
MY.NET.12.6     2939
MY.NET.60.17    28
MY.NET.34.14    20
MY.NET.97.21    15
MY.NET.34.11    15

------------
Top 5 DNS:
------------
IP address      Frequency
MY.NET.1.3      305
MY.NET.1.4      87
MY.NET.1.5      63
MY.NET.18.25    16
MY.NET.97.55    11

------------
Top 5 Web:
------------
IP address      Frequency
MY.NET.30.4     757
MY.NET.53.84    525
MY.NET.17.4     458
MY.NET.24.44    396
MY.NET.17.3     396
```

The results indicate that there are 15,747 different internal IP addresses (823 if the scan logs are not used). It is probably a B class network —with or without subnetting— or a set of consecutive C class networks, because the first and second byte of the IP addresses are *assumed* to be equal (`MY.NET`), while the third byte has several values. The third byte of the IP addresses ranges from 1 to 191, but some numbers are not present (like 3, 8, 19, 23...)

On the one hand, it is possible that C class networks have been assigned to different departments of the University. The reason of having some C class subnetwors that are not present in this list could be because they have been allocated for future use —departments that might have more systems in the future,— assigned to departments that are not currently using them, or simply because no activity related to them has been detected.

On the other hand, it is also possible that it is a class B network. If so, subnetting could have been used to make administration of the different parts of the network easier. There are IP address in the alerts in which the value of the last byte is 0. These can be valid addresses in a class B network with no subnetting or with a division of the network with a mask that has less than 24 bits. It is also possible that these alerts correspond to either ill formed target addresses or spoofed source addresses.

The script also finds the systems that have more activity associated with the most common well known ports. From its results, it seems that MY.NET.12.6 is a

mail server, because it has generated a lot of entries —many more that the next one in the list— in the logs that involve this kind of activity. The same reasoning can be applied to identify MY.NET.1.3 as a name server. However, there is no clear winner for the web server role. It is possible that the top five web systems printed by the script are all running a web server.

Figure 1 is a network diagram that conforms the previous descriptions.

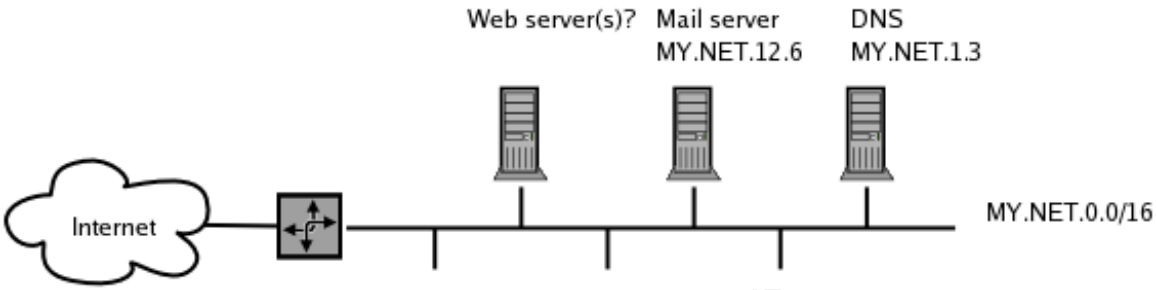

Figure 1: Network diagram

For the link graph, figure 2, I decided to represent the activity that is studied in the detect 2.

3 Detects

I started reviewing the alert logs, and counting the number of entries —one per line— in each file:

```
# wc -l alert.04042*
 112319 alert.040420
 221462 alert.040421
 218187 alert.040422
```

The purpose of this review is to filter out the less relevant entries of the log files. I started using only a combination of `less` —a common Linux program to scroll through texts,— `grep` —a program to extract strings from text files,— and `wc` —a utility to count the lines, words and characters of a tex file. However, after some tries I decided to use Perl also, because it would offer me all these functionality and some more in an integrated script.

Using `less` I opened the file `alert.040420`. All but one of the first fourty lines of the file are described as “spp_portscan: ”. Portscanning is part of the information gathering that an attacker should do during the scanning phase *before* starting the actual attack. Only the IP address address of the system that originated the portscan is stored with the alert, so I did not have in the alert logs any information about the target machine(s), the ports they have open, nor how many replies have been sent from the systems that identify the ports as open or closed. For all these reasons, I decided to concentrate on the rest of the alerts and ignore the portscans, at least for a while. I then used `grep` to verify that the most representative string for those alerts (portscan) can be used to filter them and only them out.

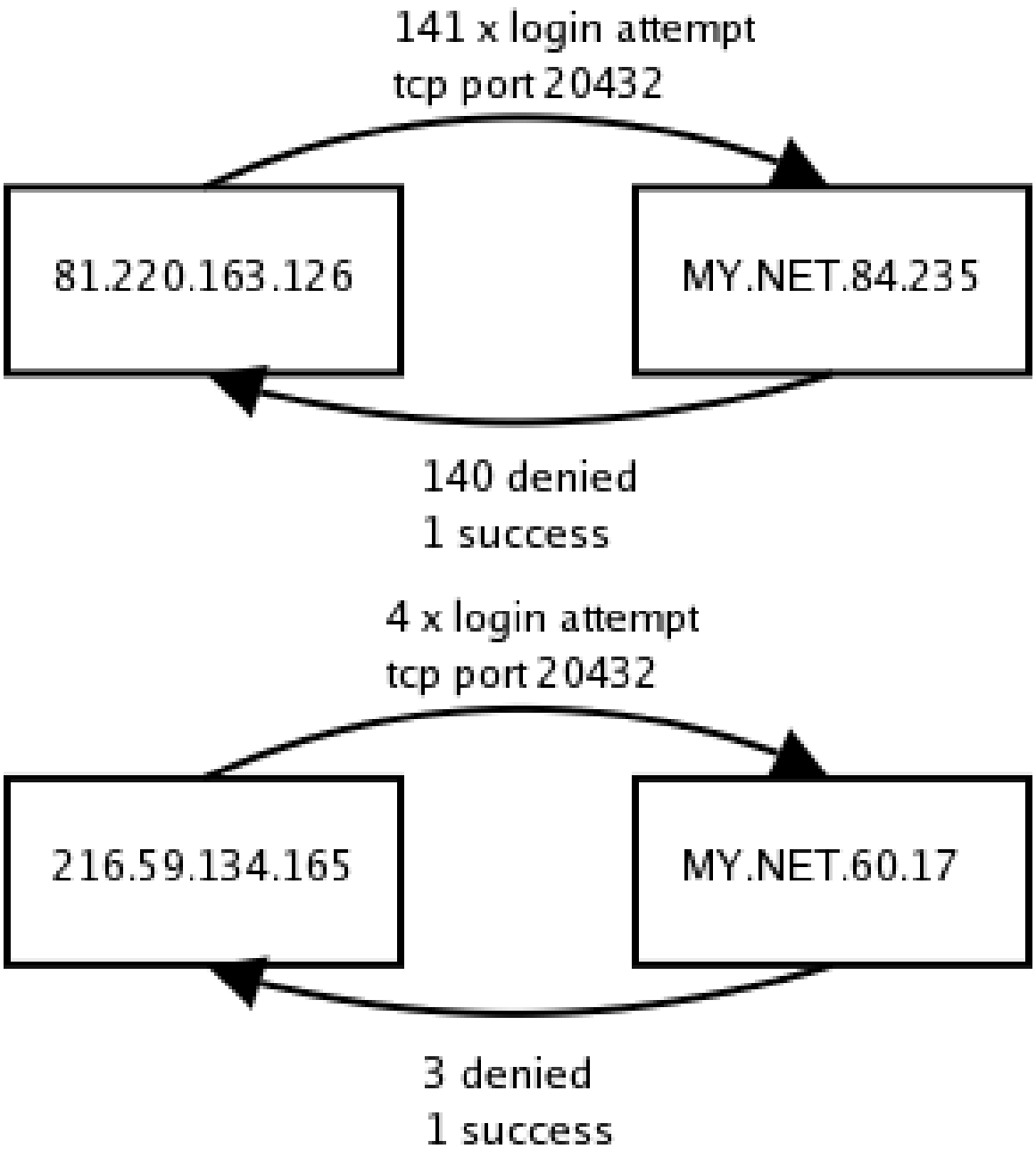

Figure 2: Link graph

```
grep spp_portscan alert.040420 | less
```

I took a look at the results from the previous command and verify that only spp_portscan alerts had been selected. I then used `grep` again to check if there was any other alert about portscanning different than the ones selected before.

```
grep spp_portscan alert.040420 | grep -i portscan | wc -l
      0
```

I included this filter in the Perl script that can be found in section 9.2 (lines 12–13) and went on with the rest of the entries.

```
filter_alert.pl alert.040420 | less
```

From this point on I followed the process described above with the three alert files to discard other entries. The main loop of this process can be summarized as follows:

1. Use `less` to review the remaining entries —the current output of the Perl script— looking for entries that appear frequently but seem unimportant.
2. Choose an expression that is part of the entry, so it can be used to filter out all those and only those entries.
3. Use `grep` to verify that only the selected expressions are filtered.
4. Optionally use `wc -l` to determine how many entries are going to be filtered.
5. Include the expression in the Perl script to filter out and count those entries.

After some work I have decided to filter out several types of alerts. Below there is a list with the types of alerts that I decided to filter out and the reason why I did so.

***IP address* activity** These are custom alerts to detect activity directed to one of these two machines: `MY.NET.30.3` and `MY.NET.30.4`. These two machines are probably acting as honeypots[2]. All this activity is originated from other IP addresses —as can be seen in the output of the script— and there is no alert that has been produced by those two systems.

NMAP This matches both Nmap tcp pings and nmap fingerprinting attemptes, which are the part of the scanning phase too. They are performed before or during the course of a portscan as part of the information gathering process. They determine if a target host is alive and which operating system is installed in the target box.

Null scan This is another type of scan implemented by Nmap as described in its man page[3].

SYN-FIN scan This is yet another type of scan implemented by Nmap as described in its man page[3].

SMB Name Wildcard or **NETBIOS NT NULL** These are part of the scanning phase too and it is done as part of the information gathering process to obtain NetBIOS information[4].

SRC and DST These alerts are generated either because of routing/filtering problems or because of packets crafted inside of the University's network that contain a spoofed source address. I would pass them to the network administrator to have them checked.

TCP SMTP Source Port traffic This alert identifies traffic that is originated from the SMTP source port, that is 25. However, this is normal when two mail servers interchange emails. It seems that MY.NET.12.6 has been previously identified as a mail server. The administrators should confirm if this is the authorized mail server.

Possible trojan server activity This is probably a custom alert created to detect traffic to or from TCP port 27374 —and some others— that is commonly used by SubSeven and other trojans (Bad Blood, DefCon 8, etc.)[8]. As in the previous case, there are several possible false positives because this port is also used as an ephemeral port.

DDOS mstream client to handler This alert has the purpouse of detecting mstream traffic. Clients communicate to mstream handlers through port 12754 as explained in [9]. This is also an ephemral port, which can generate false positives.

SUNRPC highport access This alert tries to detect access to Sun RPC through port 32271. As in the previous cases, this can also be an ephemeral port. I have filtered out in the script all the instances of this alert that record traffic to a well known port (smtp, http, identd and AIM).

RFB - Possible WinVNC This alert is generated when traffic from a Windows VNC server —a remote desktop application similar to Microsoft's Terminal Server— which could be the sympthom of a compromised system or unauhtorized use of VNC to control any of the systems of the University from outside of the perimeter.

The result of filtering all those alerts out with the aforementioned Perl script, generates the folloing output:

```
# filter_alert.pl alert.04042* > /dev/null
Alert type      Frequency
portscan        464610
honeypots       40118
red worm        19360
exploit x86     11212
smb wildcard or null session    5979
fragment        4452
winvnc  2358
```

```
null scan       1728
nmap    629
trojan server   362
sunrpc highport 240
irc     194
shaft   145
src and dst     120
smtp    97
invalid 82
ftp passwd      60
smb c access    40
external tftp   37
other rcp       32
internal tftp   17
mimail  16
nimda   15
ftpd globbing   15
dameware trojan 12
ddos mstream    11
exploit ntpdx   9
mstream 6
external spooler        4
syn-fin scan    3
external ftp helpdesk   3
anomalous       1
back orifice    1

Honeypots:
MY.NET.30.3 => 9269
MY.NET.30.4 => 30849
COMPROMISED: 0
```

This output shows how many alerts of each type have been filtered out. Nevertheless, there are still 35,631 alerts left out of the initial 551,968 alerts —less than a 6.5%,— which should be analyzed more carefully. I have selected three different types of those that I consider most interesting —for the reasons explained below— to analyze them in more detail.

3.1 Detect 1: Resurrection of the Adore worm?

3.1.1 Description of detect

```
04/20-14:00:05.100064  [**] High port 65535 udp - possible Red Worm - traffic [**]
 66.250.188.23:32767 -> MY.NET.66.29:65535
04/20-14:02:49.789800  [**] High port 65535 udp - possible Red Worm - traffic [**]
 212.187.204.47:65535 -> MY.NET.10.12:65535
04/20-14:03:17.743390  [**] High port 65535 udp - possible Red Worm - traffic [**]
 212.187.204.47:65535 -> MY.NET.10.12:65535
04/20-13:54:19.309575  [**] High port 65535 tcp - possible Red Worm - traffic [**]
 MY.NET.24.34:80 -> 66.194.21.200:65535
04/20-13:54:19.313665  [**] High port 65535 tcp - possible Red Worm - traffic [**]
 MY.NET.24.34:80 -> 66.194.21.200:65535
04/20-13:54:19.313807  [**] High port 65535 tcp - possible Red Worm - traffic [**]
 MY.NET.24.34:80 -> 66.194.21.200:65535
  :
```

This activity is associated with the Adore Worm, previously known as Red Worm. This worm targets Linux i386 systems, mostly Red Hat 7.0 but other Linux i386 systems with the same vulnerabilities will also be compromised. It uses four previously known vulnerabilities to distribute itself to other systems.

Once a system is compromised it downloads a tar file from go.163.com and installs its contents. It hides itself with a trojanized version of `ps`, enables anonymous FTP access, and launches a backdoor that gets activated using ICMP and listens to port 65535 both TCP and UDP. After notifying by email that the system has been compromised it tries to distribute itself to other systems using the following vulnerabilities:

- A remote format string vulnerability in wu-ftpd[12] (CVE-2000-0573).
- A remote format string vulnerability in rpc.statd[13] (CVE-2000-0666).
- A remote format string in LPRng[14] (CVE-2000-0917).
- A remote buffer overflow in BIND[15] (CVE-2001-0010).

A very detailed explanation of the Adore worm can be found in [10] and [11].

3.1.2 Reason this detect was selected

I considered very interesting this activity because the original Adore worm requiered a file to be downloaded from a site that was closed for a period of time because of the alarm generated by this attack. I would have thought that the mechanism that the worm uses to propagate itself was not efective anymore by the time this alerts were captured. However, there are a lot of alerts registering this kind of activity in the logs captured in April 2004 as shown with the following command:

```
# grep "Red Worm" alert.04042* | wc -l
  19381
```

3.1.3 Detect was generated by

The detect was generated by one of the intrusion detection systems of the University. This system is running Snort. The rules that were triggered are not present in my rulebase, but it seems that they look for any traffic directed to TCP or UDP port 65535. A possible reconstruction of the rules would be:

```
alert tcp any any <> any 65535 \
(flow: established; \
msg: "High port 65535 tcp - possible Red Worm - traffic";)
alert udp any any <> any 65535 \
(msg: "High port 65535 udp - possible Red Worm - traffic";)
```

I have not been able to determine, from the documentation about the Adore Worm that was mentioned previously, if there is some characteristic content in the packets. The raw capture packets would be very helpful for this task.

If the rules are similar to the ones above, they would be triggered by false positives. Actually, when I first saw the entries, I thought that it was a false positive: a web server talking to an ephemeral port. But then I realized that there are a few UDP entries also and it is not very common to see traffic from UDP port 80.

Table 1 contains, in the first column, the external IP addresses that have generated more Red Worm alers. The second column is the number of Red Worm alerts that have been generated by that IP address. The third and the fourth column are the number of alerts due to TCP and UDP traffic respectively. And the last two columns are the number of portscans and alerts, other than Red Worm alerts, that have been generated by the corresponding IP address.

The table does not include every IP address, but only the ones that generated more alerts. The Perl script included in section 9.3 can be used to generate an exhaustive list of IP addresses together with their correspondent data.

The top five IP addresses in table 1 —and probably many others— belong to ISPs (America Online, Comcast Cable Communications, T-OnlineFrance, and UUNET Technologies).

IP address	Red Worm	RW tcp	RW udp	Portscans	Other Activity
64.12.24.34	6140	6140	0	4	1
67.167.3.240	4504	4504	0	1	1
64.12.24.35	4495	4495	0	4	3
195.36.245.141	2231	2231	0	3	1
65.222.188.7	344	344	0	0	0
64.12.201.10	312	312	0	0	0
202.171.70.94	141	141	0	0	0
64.12.30.172	128	128	0	0	0
204.235.236.35	120	120	0	0	0
218.30.19.116	85	85	0	0	0
205.188.5.100	85	85	0	0	0
212.195.185.226	65	65	0	0	0
64.136.109.8	57	57	0	0	1
192.35.35.36	56	56	0	0	0
205.188.7.64	39	39	0	0	0
205.188.179.73	38	38	0	0	0
202.171.71.17	28	28	0	0	0
81.5.168.8	21	0	21	0	0
219.111.73.192	18	18	0	0	0
216.148.227.126	17	17	0	0	0
64.12.26.32	17	17	0	0	0
63.110.173.3	17	17	0	0	0
64.12.161.153	16	16	0	0	0
12.158.35.251	15	15	0	18	0
:	:	:	:	:	:

Table 1: External IP addresses involved in Red Worm alerts.

3.1.4 Probability the source address was spoofed

It seems very unlikely that most source addresses are spoofed since many of them correspond to stablished TCP sessions. There is still chance of them to be spoofed if the systems involved have good predictability of the IP sequence number.

3.1.5 Attack mechanism

In this detect the alert was triggered to indicate that those systems were being accessed through a backdoor that had been supposedly installed by the Adore worm.

As I explained before, the Adore worm downloads a file from go.123.com to install some binaries in the system as part of its distribution mechanism. This site was taken off-line three years ago and the required file has been supposedly unavailable since then. So an alternate explanation for this traffic is required.

There are three possible explanations for it:

1. They are false positives.
2. These systems have been infected for a very long period of time, the backdoor is still available and someone knows it.
3. This is a new variant of the Adore worm or another malicious software that still uses the same backdoor or, at least, the same access ports.

The first one would imply that normal traffic —like the connections from an ephemeral port— triggers this alert. This seems very unlikely considering the number of alerts and that some of the systems have generated a big number of them from many different connections.

The second one is not very probable either. There is a big number of systems that would have been running the backdoor for the last three years. Although, Linux systems are very stable most environments upgrade their i386 systems more often. System administrators should verify whether the internal systems have been reinstalled since then or not.

The third one is the most plausible. The amount of traffic directed to TCP port 65535 is unusualy big.

If the systems were infected with the Adore worm they would try to propagate using the aforementioned vulnerabilities. Despite the big number of alerts about `exploit x86`, that would capture at least explotation attempts against the buffer overflow vulnerabilities, I have been unable to find any activity that relates to the Adore worm distribution process. The activity in the alert, oos and scan logs that involves connections to the ports that run the vulnerable services (21, 53, 111, and 515) has no relation with any of the systems that generated the Red worm alerts.

I would recommend that one of those internal systems is investigated by one member of the security staff together the system administrator.

3.1.6 Correlations

This activity has been detected in previous practicals such as[16] and [17].

3.1.7 Evidence of active targeting

Most systems involved in this traffic do not show up in almost any other alerts as can be verified in the output of the Perl script partially shown in table 1. Only the

top four have been registered in other alerts and portscans. Probably those four systems are compromised or, more rarely, belong to an attacker.

This traffic is directed to a big number of internal hosts —more than 50— which would be normal behaviour for a worm.

3.1.8 Severity

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

severity = (3 + 5) - (1 + 2) = **5**

Criticality I do not know the function of the infected systems so I will take the middle value. Criticality is 3.

Lethality This is probably a backdoor, that is already installed in many systems using an unknown distribution method. Lethality is 5.

System Countermeasures The affected systems have the backdoor already installed. System countermeasures is 1.

Network Countermeasures The backdoor is being accessed from the outside, although it is a high port not assigned to a well known service. Fortunately the presence of IDSs sensors has helped to detect this traffic. Network countermeasures is 2.

3.2 Detect 2: Knocking on System's (back)Door

3.2.1 Description of detect

```
04/20-14:45:09.171005  [**] DDOS shaft client to handler [**]
 81.220.163.126:4662 -> MY.NET.84.235:20432
04/20-14:42:38.799893  [**] DDOS shaft client to handler [**]
 81.220.163.126:4662 -> MY.NET.84.235:20432
04/20-14:42:40.160566  [**] DDOS shaft client to handler [**]
 81.220.163.126:4662 -> MY.NET.84.235:20432
04/20-14:42:40.778421  [**] DDOS shaft client to handler [**]
 81.220.163.126:4662 -> MY.NET.84.235:20432
  :
```

This activity is associated with *Shaft*. Shaft is a distributed denial of service (DDoS) tool. The systems that generate the traffic that exhausts the resources of the target system have an agent installed. These agents are commanded by one or more handlers that provide one more level of indirection and make possible the control of a big number of agents. The client sends commands to every handler using TCP port 20432. Communication between the handler(s) and each agent is done through two udp ports.

All the DDOS shaft alerts refer to a client communicating to a handler.

Very detailed explanations about the way Shaft clients, handlers and agents work can be found in [5], [6] and [7].

3.2.2 Reason this detect was selected

I decided to investigate these alerts because the University could be liable for any attacks to Web sites that had been originated here.

It is also interesting that a DDoS tool that was created in 2000 can still used without even modifying the ports used. The problem with this is that chances are good that the traffic gets detected. By their own nature, the agents and the handlers can be distributed among many different networks and any of them can detect the traffic and notify the others. Having just one agent in a network that has a the right signatures in place to detect Shaft traffic could cause the detection of all the systems and the dismantlement of the DDoS network.

3.2.3 Detect was generated by

The detect was generated by one of the intrusion detection systems of the University. This system is running Snort. The rule that was triggered is still present in my rulebase and probably in the same version that I have since this signature has not changed a lot lately. The current snort rule is:

```
alert tcp $HOME_NET 20432 -> $EXTERNAL_NET any \
(msg:"DDOS shaft client login to handler"; flow:from_server,established;\
 content:"login|3A|"; reference:arachnids,254;\
 reference:url,security.royans.net/info/posts/bugtraq_ddos3.shtml;\
 classtype:attempted-dos; sid:230; rev:5;)
```

This rule does not only detect a tcp connection to port 20342 in one of the systems of the University, but it also checks that:

- The connection has been established.
- The internal system —the one using port 20432— is acting as the server.
- The packet contains a *login:* prompt.

Snort Signature Database[18] mentions the possibility of false positives if a legitimate server is using this port or if its used as a data port during an FTP session.

There is no CVE number related to this because having a DDoS tool installed in the system is not a vulnerability itself, but the result of the explotation of *any* vulnerability by an attacker.

3.2.4 Probability the source address was spoofed

This rule is only triggered if the TCP session has already been stablished. Thus, it is very unlikely that the external IP addresses have been spoofed. There is still chance of them to be spoofed if the systems involved have good predictability of the IP sequence number.

3.2.5 Attack mechanism

If the system is already acting as a Shaft handler it means that it was previously compromised. If it is running as a Shaft agent the intruder most probably got root access so it can generate raw IP packets.

The communication is used to instruct the handler to command the agents or to perform different operations related to DDoS attacks. A very detailed explanation about the possible commands can be found here[5].

No traffic from the handler to the agents has been detected in the logs, hopefully because it did not ocurr.

Table 2 contains in the first column, the IP addresses that were involved in the DDOS shaft alers. The second column is the number of Shaft alerts that have been generated by that IP address. The last column is the number of alerts, other than DDOS shaft alerts, that have been generated by the corresponding external IP address. This data can be generated using the Perl script included in section 9.4.

Internal		
IP address	DDOS shaft	
MY.NET.84.235	141	
MY.NET.60.17	4	
External		
IP address	DDOS shaft	Other activity
81.220.163.126	141	0
216.59.134.165	4	0

Table 2: IP addresses involved in DDOS shaft alerts.

It is very significant that only four IP addresses, two as handlers and two as clients, are involved in this alerts. Futher looking at the alerts reveals that the 141 alerts, corresponding to the first pair of systems, have been generated in 31 seconds while the other 4 alerts were generated within 5 minutes.

Since every packet that generates an alert contains a `login:` prompt the first conversation looks like the session of a dictionary attack, with at least 141 common passwords for shaft handlers. There is no way of knowing if the attack was successful from the three types of logs. A full network capture would be required for that. What seems to be certain is that there is a server —possibly a shaft handler— listening to port 20432 in MY.NET.84.235.

The other conversation is not so clear. The first two alerts are generated in less than a milisecond. The same happens with the other two, but both pairs are 5 seconds away. It looks like the IDS is getting two copies of each packet for this conversation. The client tries to authenticate once and fails, and it succeeds the next time.

3.2.6 Correlations

There are alerts available to detect this traffic from both the ArachNIDS site[6] and the Snort Signature Database[18].

Similar activity has been detected in previous GCIA practicals such as [19] and [20].

3.2.7 Evidence of active targeting

This traffic targets very specific hosts. Probably the first target was found as a result of a previous portscan —although no evidence has been found in the logs that have been analyzed,— while the other one was previously known by the person connecting to this system.

3.2.8 Severity

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

severity = (3 + 4) - (1 + 2) = **4**

Criticality I do not know the function of the infected systems so I will take the middle value. Criticality is 3.

Lethality They are probably two instances of a DDoS handler, installed because the system were previously compromised. Lethality is 4.

System Countermeasures The affected systems have the DDoS tool already installed. System countermeasures is 1.

Network Countermeasures The DDoS tool is being accessed from the outside, although it is a high port not assigned to a well known service. Fortunately the presence of IDSs sensors has helped to detect this traffic. Network countermeasures is 2.

3.3 Detect 3: Wrong neighborhood

3.3.1 Description of detect

```
04/20-22:13:37.185606  [**] connect to 515 from inside [**]
 MY.NET.97.186:3609 -> 192.168.2.1:515
04/20-22:13:40.226470  [**] connect to 515 from inside [**]
 MY.NET.97.186:3609 -> 192.168.2.1:515
04/20-22:14:03.134777  [**] connect to 515 from inside [**]
 MY.NET.97.186:3610 -> 192.168.2.1:515
04/20-22:14:06.112025  [**] connect to 515 from inside [**]
 MY.NET.97.186:3610 -> 192.168.2.1:515
```

This activity is probably detected by a custom alert created to discover any leak of information —produced either by an insider that sends information to somebody else or an intruder that has already compromised the system— as well as any try to exploit one of the several vulnerabilities —like the one used by the Adore worm [14]— present in the various printer spooler daemons (LPR or LPRng) like CVE-2000-0917.

Any traffic to a printer spooler daemon outside of the perimeter is considered suspicious because it is not very common for people to want to print something outside of the facilities where they are currently working.

3.3.2 Reason this detect was selected

I selected this detect because I thought it would be interesting to analyze a case that could involve a leak of information. This kind of issues are getting more and more relevant with the increasing awareness on intelectual property threats these days.

3.3.3 Detect was generated by

The detect was generated by one of the intrusion detection systems of the University. This system is running Snort. The rules that were triggered are not present in my rulebase, but it seems that they look for any traffic directed to TCP port 515. A possible reconstruction of the rule would be:

```
alert tcp $HOME_NET any <> $EXTERNAL 515 \
(msg: "connect to 515 from inside";)
```

All the alerts came from the same IP address and the destination IP address is also the same one.

The rule does not inspect the packets for any content nor checks whether the flow is stablished.

3.3.4 Probability the source address was spoofed

It is possible that the IP address was spoofed, since, as I previously stated the rule does not check if the flow is stablished. However, if they are false positives as I explain below it would make no sense, since there would be nothing to hide.

3.3.5 Attack mechanism

If the rule is similar to the one above, it would be triggered by false positives. Actually, it seems that the four alerts that are present in the alert logs are in fact a false positive, because the destination address is 192.168.2.1, which is a private address as stated in the RFC-1918[21] and should be rejected by external routers.

Three possible explanations of these four alerts come to my mind:

- An authorized private newtwork exists and is not declared as part of the internal network in the snort configuration file.
- An unauthorized private network exists and static routes have been declared in MY.NET.97.186 so it can access that network. The static route goes through the segment in which the IDS is installed.

- MY.NET.97.186 has a configuration file with non valid —at least, when connected this network— or outdated information. This system could even be a portable computer that has a printer with this IP address configured for another network that it has been connected to.

The following facts can be extracted from a closer inspection of the alerts:

- There are only two source ports present in the four alerts. The source port in first two alerts is 3609 and in the last two is 3610.
- The first and the second alert are 3 seconds apart. The third and fourth alerts are also 3 seconds apart. The first and second pair are 23 seconds apart.

Three seconds is a common defaul value for the initial retrasmission timeout of a connection retry in TCP. If the user kills the process no other packets are sent. They look like connection retries: the user tries to print and 4–8 seconds later kills the process because it does not work. Twenty-three seconds later, after reviewing its configuration and thinking that everything seems fine, he tries to print again. 4–8 seconds later he realizes that the system is trying to connect to a wrong address and kills the process again. Hopefully he fixed the problem and printed successfully after that.

Thus, they can be dismissed as connections retries. A full network trace would confirm if the connection was stablished at anytime.

3.3.6 Correlations

Similar alerts have been analized in previous practicals, such as [22] or [23]. However, the result of the analysis was completely different as they were associated to malicious traffic trying to exploit vulnerabilities in the priter spooler as the one described in [14].

Other practicals have also detected private addresses in the traffic and associated it with configuration problems, like [24].

3.3.7 Evidence of active targeting

The system that generated the alert did not show up in any other entry of the alert, oos, and scan logs. Two other —SMB Name Wildcard— alerts were generated with the same target, probably due to the same misconfiguration problem in any other system.

3.3.8 Severity

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

severity = (1 + 1) - (3 + 4) = **-5**

Criticality This was not a real attack and the main reason for triggering the alert was that the target system did not exist. Criticality is 1.

Lethality There was no damage to the target system since it did not exist. Lethality is 1.

System Countermeasures No especial system countermeasures are required nor in place for this detect. I think that chosing the middle value is a fair decision, indicating that the system did not have more or less countermeasures in place that those that were required. System countermeasures is 3.

Network Countermeasures Private networks are not being routed in the external routers and that is way —among other things— it did not get an answer. Network countermeasures is 4.

4 Network Statistics

All the statistics presented in this section are obtained using the Perl script that can be found in section 9.5.

Table 3 contains the top five talkers together with the number of times this IP addresses appear in the logs.

IP address	Frequency
MY.NET.1.3	2640587
MY.NET.17.45	1191898
MY.NET.1.4	831375
MY.NET.81.39	765374
MY.NET.112.189	737455

Table 3: Top five talkers.

Table 4 contains the list of the five top targeted ports. This list is the result of counting the number of times each port is registered in the logs, but *only as a destination port*. Counting also the source ports would generate a very different table that contains ephemeral ports in the top five list.

Port	Frequency
53	3273340
135	1847006
80	540341
2745	495672
6129	371258

Table 4: Top five target ports.

Table 5 contains the three most suspicious external source addresses based on the amount of activity captured in the logs.

Whois returns this information for the most suspicious IP addresses:

Suspect	Frequency
213.180.193.68	39515
220.197.192.39	31259
64.136.199.197	23704

Table 5: Top three suspicious external IP address.

```
whois 213.180.193.68
[Querying whois.ripe.net]
[whois.ripe.net]

inetnum:      213.180.192.0 - 213.180.193.255
netname:      COMPTEK-NET1
descr:        CompTek International/Yandex LLC
descr:        3, Gubkina str., Moscow, 117809
country:      RU
  :

role:         Yandex LLC Network Operations
address:      Yandex LLC
address:      40A Vavilova st.
address:      117333, Moscow, Russia
  :
```

```
# whois 220.197.192.39
[Querying whois.apnic.net]
[whois.apnic.net]

inetnum:      220.192.0.0 - 220.207.255.255
netname:      UNICOM
descr:        China United Telecommunications Corporation
descr:        No.133,Taiyun Building,Xidan North Street
descr:        Xicheng District,Beijing,China
country:      CN
  :

role:         Unicom China Hostmaster
address:      911 Room,Xin Tong Center,No.8 Beijing Railway Station
address:      East Avenue, Beijing,PRC.
country:      CN
  :
```

```
# whois 64.136.199.197
[Querying whois.arin.net]
[whois.arin.net]
Everest Connections, LLC EVEREST-BLK3 (NET-64-136-192-0-1)
                                64.136.192.0 - 64.136.223.255
Everest Broadband EVEREST-KSLECMTS-2 (NET-64-136-192-0-2)
                                64.136.192.0 - 64.136.207.255

# ARIN WHOIS database, last updated 2004-11-14 19:10
```

Google finds many web pages about Everest Connections, an Internet provider based in Kansas City.

It would be good to verify with the system staff if there is a legitimate reason for people connecting from China or Russia.

I could not obtain any information about the operating systems installed in those machines because there were no raw captures available.

5 Correlations

I have previously stablished correlation with other GCIA practicals: [16], [17], [19], [20], [22], [23], and [24].

Sans Top 20[25] also mentions BIND as the most commonly exploited UNIX/Linux application because of its different vulnerabilities such as the one used by the Adore worm (CVE-2000-0010).

6 Malicious activity

Malicious activity has been detected in two of the three detects. Many systems could be running backdoors and some others could be running DDoS tools. Fortunately none of this systems acts as the system responsible for the main services offered at the University.

The Chief Security Officer (CFO) should initiate actions so the affected systems are investigated and recovered to a *clean* state. Other events should be investigated as well as recommended previously.

7 Recommendations

In order to solve or mitigate the problems shown here, I would recommend doing the following things:

- Increase security awareness among the alumni, faculty, and staff.
- Create a computer emergency response team that is responsible for investigating suspicious activity.
- IDS should be managed more carefully. The rulebase should be updated often and the configuration customized to reduce the number of false positives.
- There should be people trained to conduct periodic revisions of the IDS alerts, reporting suspicious activity to the computer emergency response team.
- If possible for this University install a perimeter firewall. If it is already installed configure it properly: deny by default.
- Capture events from other systems as well: the firewall, the routers, and the critical systems at least. Time should be syncronized in order to make sense of the results of the correlations of logs.
- I would recommend keeping a full network capture for as long as it is possible. If this is impossible due to the storage requirements or the protection of the privacy of the users, I would at least change the alert mode to full in all cases.

- Much of the activity detected can be related to old vulnerabilities. Systems —at least the ones administered by the University staff— should be patched short after the patch for critical vulnerabilities has been released.
- Stablish or review the system configuration policy. Require personal firewalls, antivirus, and safe defaults.

Part III
Analysis Process (20 points)

8 Analysis Platform

The analysis of the data has been done in Pentium 4 workstation with 1 GB RAM and 120 GB hard disc, running Fedora Core 1 Linux. The packages described in table 6 were installed in the system.

Package name	Description	Version
bash	Bourne Again shell command language interpreter	2.05b-34
perl	Perl programming language	5.8.3-16
coreutils	GNU core utils (wc, ...)	5.0-34.1
less	A text file browser that resembles more	382-1.1
grep	Utility to search for matching lines through textual input	2.5.1-17.4
tcpdump	Command-line tool for monitoring network traffic	3.7.2-8.fc1.2
snort	Open source network intrusion detection system	2.2.0-0.fdr.1

Table 6: Software packages installed in the analysis workstation.

9 Perl scripts

9.1 Topology

The following script is used to extract IP addresses from the log files and calculate the number of external and internal addresses. It also looks for system activity related to three very common ports in order to identify the mail server(s), the name server(s), and the web server(s).

topology.pl

```
1 #!/usr/bin/perl
2
3 foreach $arg (@ARGV) {
4     open(FILE, $arg) || die "Couldn't open file $arg";
5     while ($line = <FILE>) {
6         # portscan entries
7         if ($line =~ /portscan.* from (\w+\.\w+\.\d+\.\d+)/i) {
8             $ip = $1;
```

```
            $ip =~ s/^130\.85\./MY.NET./;
            $address{$ip} = 1;
            if ($ip =~ /^MY\.NET\./) {
                $internal{$ip} = 1;
            } else {
                $external{$ip} = 1;
            }
        # ICMP entries
        } elsif ($line =~ /(\w+\.\w+\.\d+\.\d+) -> (\w+\.\w+\.\d+\.\d+)/) {
            $ip1 = $1; $ip2 = $2;
            $ip1 =~ s/^130\.85\./MY.NET./;
            $ip2 =~ s/^130\.85\./MY.NET./;
            $address{$ip1} = 1;
            $address{$ip2} = 1;
            if ($ip1 =~ /^MY\.NET\./) {
                $internal{$ip1} = 1;
            } else {
                $external{$ip1} = 1;
            }
            if ($ip2 =~ /^MY\.NET\./) {
                $internal{$ip2} = 1;
            } else {
                $external{$ip2} = 1;
            }
        # any other entry
        } elsif ($line =~ /(\w+\.\w+\.\d+\.\d+)\:(\d+) -> (\w+\.\w+\.\d+\.\d+)\:(\d+)/) {
            $ip1 = $1; $ip2 = $3;
            $port1 = $2; $port2 = $4;
            $ip1 =~ s/^130\.85\./MY.NET./;
            $ip2 =~ s/^130\.85\./MY.NET./;
            $address{$ip1} = 1;
            $address{$ip2} = 1;
            if ($ip1 =~ /^MY\.NET\./) {
                $internal{$ip1} = 1;
                if ($port1 == 25) {
                    $mail{$ip1}++;
                } elsif ($port1 == 53) {
                    $dns{$ip1}++;
                } elsif ($port1 == 80) {
                    $web{$ip1}++;
                }
            } else {
                $external{$ip1} = 1;
            }
            if ($ip2 =~ /^MY\.NET\./) {
                $internal{$ip2} = 1;
                if ($port2 == 25) {
                    $mail{$ip2}++;
                } elsif ($port2 == 53) {
                    $dns{$ip2}++;
                } elsif ($port2 == 80) {
                    $web{$ip2}++;
                }
            } else {
                $external{$ip2} = 1;
            }
        }
    }
    close(FILE);
}

print scalar(keys(%address))." different IP addresses.\n";
print scalar(keys(%internal))." internal IP addresses.\n";
print scalar(keys(%external))." external IP addresses.\n";
print "\n------------\nTop 5 mail:\n------------\n";
print "IP address\tFrequency\n";
$i = 0;
foreach $key (sort
```

```
77              { $mail{$b} <=> $mail{$a} }
78              (keys %mail)) {
79      print "$key\t$mail{$key}\n";
80      last if ($i++ >= 4);
81  }
82  print "\n------------\nTop 5 DNS:\n------------\n";
83  print "IP address\tFrequency\n";
84  $i = 0;
85  foreach $key (sort
86              { $dns{$b} <=> $dns{$a} }
87              (keys %dns)) {
88      print "$key\t$dns{$key}\n";
89      last if ($i++ >= 4);
90  }
91  print "\n------------\nTop 5 Web:\n------------\n";
92  print "IP address\tFrequency\n";
93  $i = 0;
94  foreach $key (sort
95              { $web{$b} <=> $web{$a} }
96              (keys %web)) {
97      print "$key\t$web{$key}\n";
98      last if ($i++ >= 4);
99  }
```

topology.pl

9.2 Filtering logs

The following Perl script is used to classify the alerts in different categories and obtain the number of ocurrences for each one. It also looks for alerts that involve the honeypot systems and if there is any outgoing traffic from them.

filter_alert.pl

```
1   #!/usr/bin/perl
2
3   foreach $arg (@ARGV) {
4       open(FILE, $arg) || die "Couldn't open file $arg";
5       while ($line = <FILE>) {
6           # Filter out:
7           # invalid lines
8           if (($line =~ /^:/) || ($line =~ /^ ->/)) {
9               $entries{'invalid'}++;
10
11          # - portscans
12          } elsif ($line =~ /spp\_portscan.* from /i) {
13              $entries{'portscan'}++;
14
15          # - honeypot activity
16          } elsif ($line =~ /\[\*\*\] (MY\.NET\.\d+\.\d+) activity \[\*\*\] (\w+\.\w+\.\d+\.\d+)/) {
17              $entries{'honeypots'}++;
18              $honeypot{$1}++;
19              # If the source is the honeypot, it has been compromised
20              $compromised++ if ($1 eq $2);
21
22          # - nmap tcp pings and fingerprints
23          } elsif ($line =~ /NMAP/) {
24              $entries{'nmap'}++;
25
26          # - null scan
27          } elsif ($line =~ /Null scan/) {
28              $entries{'null scan'}++;
29
30          # - SYN-FIN scan
31          } elsif ($line =~ /SYN-FIN scan/) {
32              $entries{'syn-fin scan'}++;
33
```

```
34          # - SMB name wildcard or null session
35          } elsif ($line =~ /SMB Name Wildcard|NETBIOS NT NULL/) {
36              $entries{'smb wildcard or null session'}++;
37
38          # - routing problems or spoofed src address
39          } elsif ($line =~ /SRC and DST/) {
40              $entries{'src and dst'}++;
41
42          # - email
43          } elsif ($line =~ /TCP SMTP Source Port traffic/) {
44              $entries{'smtp'}++;
45
46          # - trojan server
47          } elsif (($line =~ /trojan server.* \w+\.\w+\.\d+\.\d+\:(\d+) -> \w+\.\w+\.\d+\.\d+\:(\d+)/)
48                  && ((($1 == 27374) && ($2 < 1024)) ||
49                      (($1 < 1024) && ($2 == 27374)))) {
50              $entries{'trojan server'}++;
51
52          # - ddos mstream
53          } elsif (($line =~ /DDOS mstream.*\] \d+.\d+.\d+.\d+:(\d+)/)
54                  && ($1 < 1024)) {
55              $entries{'ddos mstream'}++;
56
57          # SUNRPC highport
58          } elsif (($line =~ /SUNRPC highport.*\] \d+\.\d+\.\d+\.\d+\:(\d+)/)
59                  && (($1 == 25)||($1 == 80)||($1 == 119)||($1 == 5190))) {
60              $entries{'sunrpc highport'}++;
61
62          # - WinVNC
63          } elsif ($line =~ /RFB - Possible WinVNC/) {
64              $entries{'winvnc'}++;
65
66          # Alerts from here on should be investigated.
67          # - exploit x86
68          } elsif ($line =~ /EXPLOIT x86/) {
69              $entries{'exploit x86'}++;
70
71          # - exploit ntpdx
72          } elsif ($line =~ /EXPLOIT NTPDX/) {
73              $entries{'exploit ntpdx'}++;
74
75          # - miscelaneous irc activity
76          } elsif ($line =~ / IRC /) {
77              $entries{'irc'}++;
78
79          # - Red worm
80          } elsif ($line =~ /Red Worm /) {
81              $entries{'red worm'}++;
82
83          # - DDOS shaft
84          } elsif ($line =~ /DDOS shaft/) {
85              $entries{'shaft'}++;
86
87          # - fragmented traffic
88          } elsif ($line =~ /Fragment/) {
89              $entries{'fragment'}++;
90
91          # - Dameware remote administration
92          } elsif ($line =~ /trojan server/) {
93              $entries{'dameware trojan'}++;
94
95          # - SMB C$ access
96          } elsif ($line =~ /SMB C access/) {
97              $entries{'smb c access'}++;
98
99          # - Nimda
100         } elsif ($line =~ /NIMDA/) {
101             $entries{'nimda'}++;
```

```
102
103         # - MiMail
104         } elsif ($line =~ /MiMail/) {
105             $entries{'mimail'}++;
106
107         # - internal TFTP
108         } elsif ($line =~ /TFTP - External/) {
109             $entries{'internal tftp'}++;
110
111         # - external TFTP
112         } elsif ($line =~ /TFTP - Internal/) {
113             $entries{'external tftp'}++;
114
115         # - other rpc
116         } elsif ($line =~ /RPC/) {
117             $entries{'other rcp'}++;
118
119         # - FTP passwd attempt
120         } elsif ($line =~ /FTP passwd/) {
121             $entries{'ftp passwd'}++;
122
123         # - ftpd globbing
124         } elsif ($line =~ /ftpd globbing/) {
125             $entries{'ftpd globbing'}++;
126
127         # - DDOS mstream
128         } elsif ($line =~ /DDOS mstream/) {
129             $entries{'mstream'}++;
130
131         # - DDOS mstream
132         } elsif ($line =~ /DDOS mstream/) {
133             $entries{'mstream'}++;
134
135         # - external spooler
136         } elsif ($line =~ /connect to 515/) {
137             $entries{'external spooler'}++;
138
139         # - external ftp to helpdesk
140         } elsif ($line =~ /HelpDesk/) {
141             $entries{'external ftp helpdesk'}++;
142
143         # - Back Orifice
144         } elsif ($line =~ /Back Orifice/) {
145             $entries{'back orifice'}++;
146
147         # - anomalous traffic
148         } elsif ($line =~ /Traffic from port/) {
149             $entries{'anomalous'}++;
150
151         # Rest of the lines
152         } else {
153             print $line;
154             $entries{'LEFT'}++;
155         }
156     }
157     close(FILE);
158 }
159
160 print STDERR "Alert type\tFrequency\n";
161 foreach $key (sort
162               { $entries{$b} <=> $entries{$a} }
163               (keys %entries)) {
164     print STDERR "$key\t$entries{$key}\n";
165 }
166
167 print STDERR "\nHoneypots:\n";
168 foreach $key (sort keys %honeypot) {
169     print STDERR "$key => $honeypot{$key}\n";
```

```
}
print STDERR "COMPROMISED: ".($compromised+0)."\n";
```
filter_alert.pl

9.3 Detect 1

The following Perl script filters the alert logs looking for activity that involves any of the external IP addresses that triggered any of the red worm alerts. It also generates the list of internal and external systems that participate in the red worm alerts indicating how many times. For the external systems, it presents a separate count of the tcp and udp red worm alerts, and also shows how many portscans and other alerts has this system generated.

detect_1.pl

```
#!/usr/bin/perl

# first pass to extract the how many alerts contain the same ip address
foreach $arg (@ARGV) {
    open(FILE, $arg) || die "Couldn't open file $arg";
    while ($line = <FILE>) {
        # look only for the red worm alerts
        if ($line =~ /65535 (\w+).* Red Worm.* (\w+\.\w+\.\d+\.\d+)\:\d+ -> (\w+\.\w+\.\d+\.\d+)\:\d+/) {
            # the protocol and ips have been extracted from the alert
            $proto = $1;
            $ip1 = $2;
            $ip2 = $3;
            if ($ip1 =~ /MY\.NET\./) {
                $compromised{$ip1}++;
                $external{$ip2}++;
                if ($proto eq "tcp") {
                    $tcp{$ip2}++;
                } else {
                    $udp{$ip2}++;
                }
            } else {
                $compromised{$ip2}++;
                $external{$ip1}++;
                if ($proto eq "tcp") {
                    $tcp{$ip1}++;
                } else {
                    $udp{$ip1}++;
                }
            }
        }
    }
    close(FILE);
}

foreach $arg (@ARGV) {
    open(FILE, $arg) || die "Couldn't open file $arg";
    while($line = <FILE>) {
        foreach $system (keys %external) {
            $sys_exp = $system;
            $sys_exp =~ s/\./\\\./g;
            if ($line =~ /$sys_exp/) {
                if ($line =~ /portscan/i) {
                    $scan{$system}++;
                } elsif($line !~ /Red Worm/) {
                    $activity{$system}++;
                }
                next;
            }
        }
    }
```

```
51      close(FILE);
52  }
53
54  # print the list of internal systems in red worm alerts sorted desc by
55  # number of alerts
56  print "\n-----------\nCompromised:\n-----------\n";
57  print "IP address\tRed Worm\n";
58  foreach $key (sort
59                { $compromised{$b} <=> $compromised{$a} }
60                (keys %compromised)) {
61      print "$key\t$compromised{$key}\n";
62      $total += $compromised{$key};
63  }
64  print "TOTAL === $total\n";
65
66  # print the list of external systems in red worm alerts sorted desc by
67  # number of alerts including the number of other alerts
68  print "\n--------\nExternal:\n--------\n";
69  print "IP address\tRed Worm\tRW tcp\tRW udp\tPortscans\tOther Activity\n";
70  $total = 0;
71  foreach $key (sort
72                { $external{$b} <=> $external{$a} }
73                (keys %external)) {
74      print "$key\t$external{$key}\t" .
75          ($tcp{$key}+0)."\t".
76          ($udp{$key}+0)."\t".
77          ($scan{$key}+0)."\t".
78          ($activity{$key}+0)."\n";
79      $total += $external{$key};
80  }
81  print "TOTAL === $total\n";
```

detect_1.pl

9.4 Detect 2

The following Perl script filters the alert logs looking for activity that involves any of the external IP addresses that triggered any of the DDoS shaft alerts. It also generates the list of internal and external systems that participate in the DDoS shaft alerts indicating how many times. For the external systems, it also shows how many other alert this system has generated.

detect_2.pl

```
1   #!/usr/bin/perl
2
3   # first pass to extract the how many alerts contain the same ip address
4   foreach $arg (@ARGV) {
5       open(FILE, $arg) || die "Couldn't open file $arg";
6       while ($line = <FILE>) {
7           # look only for the red worm alerts
8           if ($line =~ /DDOS shaft.* (\w+\.\w+\.\d+\.\d+)\:\d+ -> (\w+\.\w+\.\d+\.\d+)\:\d+/) {
9               # the two ip addresses have been extracted from the alert
10              $compromised{$2}++;
11              $external{$1}++;
12          }
13      }
14      close(FILE);
15  }
16
17  foreach $arg (@ARGV) {
18      open(FILE, $arg) || die "Couldn't open file $arg";
19      while ($line = <FILE>) {
20          # check if any of the external system is involved in this alert
21          foreach $system (keys %external) {
22              $sys_exp = $system;
```

```
            $sys_exp =~ s/\./\\\./g;
            if ($line =~ /$sys_exp/) {
                # the activity should be differnt than DDOS shaft
                $activity{"$system"}++ if ($line !~ /DDOS shaft/);
                next;
            }
        }
    }
    close(FILE);
}

# print the list of internal systems in red worm alerts sorted desc by
# number of alerts
print "\n-----------\nCompromised:\n-----------\n";
print "IP address\tDDOS shaft\n";
foreach $key (sort
              { $compromised{$b} <=> $compromised{$a} }
              (keys %compromised)) {
    print "$key\t$compromised{$key}\n";
    $total += $compromised{$key};
}
print "TOTAL === $total\n";

# print the list of external systems in red worm alerts sorted desc by
# number of alerts including the number of other alerts
print "\n--------\nExternal:\n--------\n";
print "IP address\tDDOS shaft\tOther activity\n";
$total = 0;
foreach $key (sort
              { $external{$b} <=> $external{$a} }
              (keys %external)) {
    print "$key\t$external{$key}\t$activity{$key}\n";
    $total += $external{$key};
}
print "TOTAL === $total\n";
```

detect_2.pl

9.5 Statistics

The following Perl script calculates the statistics of all the files passed in the command line. It extracts every ip from the logs and if the first two bytes are 130.85, they are replaced by MY.NET —to adapt the information proceding from the scan logs,— counting every time an IP address is used in order to show the top five talkers. It also counts how many times each port is used as the destination port in the logs in order to display the top five target ports. Finally, it also displays that three external IP addresses that appear more often in the logs because they are the top thee suspicious IP addresses.

statistics.pl

```
#!/usr/bin/perl

foreach $arg (@ARGV) {
    open(FILE, $arg) || die "Couldn't open file $arg";
    while ($line = <FILE>) {
        # extract ip addresses
        if ($line =~ /(\w+\.\w+\.\d+\.\d+)/) {
            $ip = $1;
            $ip =~ s/^130\.85\./MY.NET./;
            $talker{$ip}++;
            $external{$ip}++ if ($ip !~ /^MY\.NET\./);
            $port{$1}++ if ($line =~ /-> \w+\.\w+\.\d+\.\d+\:(\d+)/);
        }
    }
```

```
    close(FILE);
}

# print the list of internal systems in red worm alerts sorted desc by
# number of alerts
print STDERR "\n------------\nTop talkers:\n------------\n";
print STDERR "IP address\tFrequency\n";
$i = 0;
foreach $key (sort
              { $talker{$b} <=> $talker{$a} }
              (keys %talker)) {
    print STDERR "$key\t$talker{$key}\n";
    last if ($i++ >= 4);
}

print STDERR "\n------------\nTop targets:\n------------\n";
print STDERR "Port\tFrequency\n";
$i = 0;
foreach $key (sort
              { $port{$b} <=> $port{$a} }
              (keys %port)) {
    print STDERR "$key\t$port{$key}\n";
    last if ($i++ >= 4);
}

print STDERR "\n------------\nTop suspects:\n------------\n";
print STDERR "Suspect\tFrequency\n";
$i = 0;
foreach $key (sort
              { $external{$b} <=> $external{$a} }
              (keys %external)) {
    print STDERR "$key\t$external{$key}\n";
    last if ($i++ >= 2);
}
```

statistics.pl

References

[1] Northcut S. & Novak, J. Networ Intrusion Detection Indianapolis, IN: New Riders, 2001.

[2] The Honeynet Project Know Your Enemy Indianapolis, IN: Addison Wesley, 2001.

[3] Fyodor “Nmap network security scanner man page.” URL:http://www.insecure.org/nmap/data/nmap_manpage.html (12 Nov 2004)

[4] Whitehats “IDS177 *NETBIOS-NAME-QUERY*.” ArachNIDS. URL:http://whitehats.com/info/IDS177 (12 Nov 2004)

[5] Dietrich, S.; Dittrich, D.; Long, N. “An analysis of the *Shaft* distributed denial of service tool.” 13 Mar 2000. URL:http://www.sans.org/y2k/shaft.htm (12 Nov 2004)

[6] Whitehats “ IDS254 *DDOS-SHAFT-CLIENT-TO-HANDLER*.” ArachNIDS. URL:http://whitehats.com/info/IDS254 (13 Nov 2004)

[7] Wash, R. & Nazario, J. “Analysis of a Shaft Node and Master.” 26 Mar 2000. URL:http://biocserver.bioc.cwru.edu/~jose/shaft_analysis/node-analysis.txt (14 Nov 2004)

[8] von Braun, J. “Ports used by trojans.” 14 Mar 2001. URL:http://www.dalmatian.com/TrojanPortsfiles/nyheter9902.html (13 Nov 2004)

[9] Dittrich, D. et al. “The *mstream* distributed denial of service attack tool” 1 May 2000 URL:http://staff.washington.edu/dittrich/misc/mstream.analysis.txt (13 Nov 2004)

[10] Fearnow, M. & Stearns, W. “Adore Worm.” 12 Apr 2001 URL:http://www.sans.org/y2k/adore.htm (13 Nov 2004)

[11] Chien, E. “Linux.Adore.Worm.” 15 Apr 2002. URL:http://securityresponse.symantec.com/avcenter/venc/data/linux.adore.worm.html (14 Nov 2004)

[12] CERT/CC “CERT Advisory CA-2000-13 Two Input Validation Problems In FTPD.” 21 Nov 2000. URL:http://www.cert.org/advisories/CA-2000-13.html (14 Nov 2004)

[13] CERT/CC “CERT Advisory CA-2000-17 Input Validation Problem in rpc.statd.” 6 Sep 2000 URL:http://www.cert.org/advisories/CA-2000-17.html (14 Nov 2004)

[14] CERT/CC “CERT Advisory CA-2000-22 Input Validation Problems in LPRng.” 27 Jan 2003.
URL:http://www.cert.org/advisories/CA-2000-22.html (14 Nov 2004)

[15] CERT/CC “CERT Advisory CA-2001-02 Multiple Vulnerabilities in BIND.” 7 Aug 2001.
URL:http://www.cert.org/advisories/CA-2001-02.html (14 Nov 2004)

[16] Chuvakin, A. “GCIA Practical Assignment.” 27 Jun 2002.
URL:http://www.giac.org/practical/GCIA/Anton_Chuvakin_GCIA.pdf
(15 Nov 2004)

[17] Holstein, M. “SANS GCIA Practical Assignment.”
URL:http://www.giac.org/practical/Michael_Holstein_GCIA.doc (15 Nov 2004)

[18] Caswell, B. & Roesch, M. “Snort Signature Database.”
URL:http://www.snort.org/snort-db/ (14 Nov 2004)

[19] Clark, C. “GCIA Practical Assignment.”
URL:http://www.giac.org/practical/Crist_Clark_GCIA.html (14 Nov 2004)

[20] Van Horenbeeck, M. “Intrusion Analysis.” 8 Jan 2003.
URL:http://www.giac.org/practical/GCIA/Maarten_Vanhorenbeeck_GCIA.pdf
(15 Nov 2004)

[21] Rekhter. Y et al. “Address Allocation for Private Internets.” Feb 1996.
URL:http://www.ietf.org/rfc/rfc1918.txt (15 Nov 2004)

[22] Ellis, J. “GCIA Practical Assignment” 14 May 2002.
URL:http://www.giac.org/practical/Joe_Ellis_GCIA.doc (15 Nov 2004)

[23] Bell, M. “GCIA Practical Assignment” 14 May 2002.
URL:http://www.giac.org/practical/Mike_Bell_GCIA.doc (15 Nov 2004)

[24] Gregory, D. “SANS GIAC Intrusion Detection In-Depth.”
URL:http://www.giac.org/practical/GCIA/Donald_Gregory_GCIA.pdf
(15 Nov 2004)

[25] SANS “The Twenty Most Critical Internet Security Vulnerabilities (Updated) – The Experts Consensus.” Version 5.0 8 Oct 2004
URL:http://www.sans.org/top20/ (15 Nov 2004)