GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Network Monitoring and Threat Detection In-Depth (Security 503)"
at http://www.giac.org/registration/gcia



Intrusion Detection In Depth

GCIA Practical Assignment
Version 3.1

Frans J.H. Kollee

SANS Parliament Square 2002
London, April 22 - 27, 2002



Assignment 1 - Describe the State of Intrusion Detection - 4
Introduction. 4
Are scans worth to think about? 4
The purpose and origin of network and port scanning. 5
Should you worry about these network and port scans? 5
What can be done against these scans? 6
Obfuscating scans. 6
Honeyd - Network Rhapsody for You 7
Slowing down scans. 7
The LaBrea Tarpit 7
Other countermeasures that can be taken. 8
Conclusion 8
References 9
Additional references 9

Assignment 2 - Network Detects - 10
Detect #1 - Outbound connection attempts to a host on TCP Port 30001 10
Event traces 10
1. Source of Trace 12
2. Detect was generated by 13
3. Probability the source address was spoofed 13
4. Description of attack 14
5. Attack mechanism 15
6. Correlations 15
7. Evidence of active targeting 16
8. Severity 16
9. Defensive recommendation 16
10. Multiple choice test question 16
Detect #2 - Loadbalancing versus reconnaissance attack. 17
Event traces 17
1. Source of Trace 21
2. Detect was generated by 21
3. Probability the source address was spoofed 21
4. Description of attack 22
5. Attack mechanism 22
6. Correlations 23
7. Evidence of active targeting 24
8. Severity 24
9. Defensive recommendation 24
10. Multiple choice test question 25
Detect #3 - SNMP "public" access attempt 26
Event traces 26
1. Source of Trace 27
2. Detect was generated by 28
3. Probability the source address was spoofed 28
4. Description of attack 29
5. Attack mechanism 29
6. Correlations 31
7. Evidence of active targeting 31
8. Severity 31
9. Defensive recommendation 32
10. Multiple choice test question 32

Assignment 3 – "Analyze This"-Scenario - 33
Security Audit for "University" 33
Executive Summary 33
List of Analyzed Files 34
List of Detects 34
Analyses of the Alerts log files *34*
Analyses of the OOS logfiles *48*
Analyses of the Scan logfiles *50*
Registration Information 53
Link Graph 57
Analyses Process 59
References 60

Appendix A - Overview of all alerts - 62

Appendix B - Commands and scripts used for assignment 3 - 64
Commands01 64
Script01 65
Script02 65
Perlscript apr - alert port reporter 66

Assignment 1 - Describe the State of Intrusion Detection -

Network scans and port scans: What to do with them?

Introduction.

As soon as you have setup and implemented an Intrusion Detection System and Perimeter Device(s) as part of your company's *Security Policy*, you probably get overwhelmed by the detects that are being reported. Among these detects, you will see a lot of port scans and network scans, consuming your bandwidth and CPU time. The *Security Policy* describes what is allowed and what is not allowed. Network scans and port scans with malicious intent should not be categorized as allowed.

This paper will describe possible techniques that can be used as a defense against these overwhelming scans for vulnerable hosts and services. Network scans and host scans are often classified as annoying but not severe enough to spent much time on. This type of network traffic coming from the Internet will only increase, and it is worth to take actions upon, not only by one organization or person, but by anyone who is connected to the Internet.

From a technical perspective, there are some things that can be done against scanners by giving a little more extra efforts while implementing the company's overall security.

Are scans worth to think about?

Part of the company's *Security Policy*, will probably include an *Incident Response Cookbook* which describes how to react upon specific kind of detects that arise. At http://www.sans.org/newlook/resources/IDFAQ/deploy.htm[1], you will find a good overview about this topic which describes possible actions and provides an example of grading various incidents.

Network scans and port scans are often graded as "unfriendly" and the two major reasons for this are:

- There are a lot of these scans which have become so common and therefore are widely ignored. Investigation of every scan takes a lot of time and the results are minor according to the effort.
- The perimeter device is there to block this traffic (with success) and eventually reports upon the scans, so why spent a lot of time?

The most appropriate action is to record the source IP-address (which however could be spoofed or could be one of a compromised host) and correlate this information with other (public) sources which collect information about attackers. These public data-collections which are the result of gathering data from multiple sensors all over the world, are becoming more and more important because they can discover trends in activity.

Useful information according these data collections can be found at:

- Distributed Intrusion Detection System: http://www.dshield.org/
- Internet Storm Center: http://isc.incidents.org/
- Security Focus DeepSight Analyzer (ARIS): http://aris.securityfocus.com/
- HackerWatch.org (McAfee Personal Firewall): http://www.hackerwatch.org/

The purpose and origin of network and port scanning.

Network and port scans are often performed because of the following reasons:

- Reconnaissance and OS fingerprinting.
  The first step in hacking targets is developing an attack scheme and to learn as much as possible about the target. The final attack will be performed in regard with the previously collected information. Stealthy scans and distributed scanning are preferable techniques because they can elude an IDS. Passive OS fingerprinting is another technique used to gather information about specific hosts. Remember that hackers are willing to invest a lot of time and that there can be a huge time gap between reconnaissance and the actual hack.
- Script kiddies.
  Ready to use scripts, operated by the script kiddies are also a major source of annoying scans. These script kiddies use hacking tools downloaded from the Internet and have fun in using them, just because it's easy and exciting. They often have little knowledge about the technique and in fact they provide proof that they have no skill at all by the noisy scans.
- Virus-spreading by I-worms.
  I-worms are spreading by looking for (nearby) servers and they try to open backdoors, so hackers can gain control over these systems. Because this also is an automated process that amplifies it self, the volume of these kind of scans is enormous. Think of the different versions of Code Red. A good and detailed resource according the spreading of I-worms like Code Red is at http://www.icir.org/vern/papers/cdc-usenix-sec02/index.html [2].
- Scans originated within the internal network.
  This point is summarized as a complement. Scans with their origin within the internal network, should be treated as defined by the company's *Security Policy*. This is also true for Peer-to-Peer networks and Instant Messaging traffic with external hosts. If not explicitly allowed and monitored, you should always be aware of the malicious intent of this kind of traffic. Trojans are very likely the source of this traffic. The internal hosts are under control of the company's system administrators, so it should be easy to take appropriate countermeasures to this type of scans.

Should you worry about these network and port scans?

The answer is a definite "yes, you should" (after all, these scans are unwanted activity). The scans are not only consuming bandwidth and CPU time, but they might let you decide to loosen the rules of the IDS because of the huge amount of false positives.

The scans are often looking for exploitable services on your network and if there is one, you can be sure that someone sooner or later will find it and will take advantage of it. A successful scan which positively reports vulnerable services, isn't always followed immediately by a hack. The actual hack, can be long after the initial reconnaissance.

The summary report titled *"Internet Risk Impact Summary" for March 26, 2002 through June 24, 2002* published by Internet Security Systems available at https://gtoc.iss.net/documents/summaryreport.pdf [3] is giving some numbers about scanning. Some excerpts from this report:

- An unprotected computer will be compromised within a day of connection to the internet.
- A small but steady resurgence of the year-old Code Red.
- A known exploit for the Microsoft IIS vulnerability was noted in the wild, and an SQL worm was released which automatically exploits SQL administrator accounts that do not have a password.
- As expected Port 80, leads the list and is virtually unchanged from the previous report.
- ISS saw over half a million SQL worm events over 7500 different sources during this period.

What can be done against these scans?

Network and port scans will be on the Internet all the time and show up in different flavors. A good start upon detecting these scans is the document titled *Practical Automated Detection of Stealthy Portscans* which can be found at http://www.silicondefense.com/pptntext/Spice-JCS.pdf [4] The document describes several portscan detectors that can be used. One of these detectors is the *Snort Portscan Preprocessor*. *Spade*, which is an implementation of the *Spice anomaly detector*, is another useful preprocessor that can be used in conjunction with Snort.

Obfuscation and slowing down scans are possible countermeasures that can be done from a technical point of view. The next two paragraphs will comment on these subjects.

Blocking (shunning) the source IP is a countermeasure which is not always useful because the IP-address could be spoofed or could be randomly assigned to a customer, only for the duration of a session.

After or during the implementation of the *Security Policy*, it just takes a little extra effort to do something about the network and port scans. Whether you have only one, a couple, or a range of several C-class or B-class IP-addresses, all actions against scanning will be useful.

Obfuscating scans.

As stated before, one of the preliminary functions of scanning is OS fingerprinting. So what if you could setup a dedicated host which handles all connection-requests targeted to the unused addresses within the assigned (Internet routable) address-space and reports all kinds of operating systems or even virtual services as response to these requests?

What if you could setup an imaginary network topology which doesn't necessarily reflect the real network topology? You still would get the traffic at your site, consuming some of your bandwidth, but it could be useful as possible preliminaries to an attack yet to come and it would definitely mislead the scanner.

Honeyd - Network Rhapsody for You

The posting at http://archives.neohapsis.com/archives/sf/honeypots/2002-q2/0010.html on April 08, 2002 by Niels Provos was the announcement of *honeyd*, a small daemon that can create virtual hosts and services on a single host, providing the desired functionality as previously described. More information is available at http://www.citi.umich.edu/u/provos/honeyd/[5] which is the home of *honeyd*.

Honeyd can be used for several tasks like:

- Simulating virtual hosts by just one host, so you could assign your unused IP-address space and having them respond.
- Each virtual host can be configured to simulate arbitrary services.
- Each service can be proxied to another host.
- Simulating an operating system that fools tools like *nmap* and *xprobe*.
- Simulating routing topologies with configurable latency and packet loss.

The usefulness of this daemon is only limited by a persons imagination and it can be very useful against OS-fingerprinting. It also functions as a low profile honeynet to detect an attack that is being developed. The simulation of arbitrary services can be extended.

Slowing down scans.

Scanning for vulnerable hosts and specific services is done by using different techniques, but at high speed. Why helping the high speed scanning techniques by sending TCP-RST or ICMP messages, to notify the scanning host that he can go on with the next one?

The LaBrea Tarpit

The *LaBrea Tarpit* was developed as a means to respond to the overwhelming scans by the Code Red worm. It started with a post by Tom Liston and the basic idea of the *LaBrea Tarpit* is not to block the scanning host, but to delay the scanning-process using the design of the TCP/IP stack.

An excerpt from http://www.hackbusters.net/LaBrea/LaBrea.txt[6]

> LaBrea has the capability to capture and hold scanners - something that is of vital importance to the overall health of the Internet.
> At its peak, the Code Red worm infected approximately 300,000 servers, yet a quick "back of the envelope" calculation (note 5) indicates that 1000 sites connected to T1 lines and dedicating only 5% of their total bandwidth to LaBrea's "-p" option would have been able to capture and hold all Code Red scanning threads at once.
> And by capturing these scanning threads, LaBrea makes it possible to contact compromised system owners while keeping their systems from compromising other systems.

The basic idea of the *LaBrea Tarpit* is that it monitors ARP requests and replies. When it detects consecutive ARP-requests without replies to these requests, it responds with an ARP-reply, saying that the IP-address is at a bogus MAC-address. LaBrea than monitors inbound SYN-packets for this bogus MAC-address (0:0:f:ff:ff:ff) and responds with a SYN/ACK. Taking advantage of the connection oriented design of TCP, the *LaBrea Tarpit* responds with a small window-size advertisement (default 10) and completes the three-way TCP-handshake. The LaBrea Tarpit can response to window-size probes with a window-size of 0, meaning that the service isn't ready to receive any data and keeps the scanner busy.

There are several command-line options to configure the *LaBrea Tarpit* which tunes the running daemon like selecting the interface, setting the arp time-out, safe operation in a switched environment and more. It is also possible to define IP-addresses that must be excluded.

The home of the *LaBrea Tarpit* is at http://www.hackbusters.net/LaBrea/ and for all of them, running Windows on a cable or xDSL connection with perhaps only one assigned IP-address, there is LaBrea@home, free for personal use, at http://www.hackbusters.net/LaBrea/lbathome.html. The fact that you have been assigned only one IP-address, is no excuse not participating in the battle against scanning.

Other countermeasures that can be taken.

Besides installing and configuring separate systems, there are other, simple things that you can do against the scanners, for example:

- Configuring blackhole behavior. Some operating systems enable you to silence inbound connections (UDP an TCP) so that a scanner sends retry-packets and thus slows down
- Configuring your router to drop connections silently, the same as above but primarily for non-existing hosts.
- Changing the default values of the TCP/IP stack like TTL-value, to fool OS-fingerprinting.
- Setting up a listener that sends everything to a blackhole, for example *netcat* redirecting everything to the null-device (be careful, this could lead to a DoS).

Conclusion

A lot of companies invest a lot in setting up and deploying a *Security Policy* and probably an *Incident Response Cookbook*, but the network and port scans have become so common that people are used to it and forget about them.

There are however some simple things from a technical perspective that can be done to do something against scanners. Goal of these countermeasures is not to make the scanners disappear but to make life less easy for them.

Besides making things more difficult for scanners, bandwidth and CPU time that would have to deal with the scans, can be saved and used to perform the basic tasks.

When setting up Intrusion Detection, you should not only make scanning detectable but also take some countermeasures to react upon them. After all, scanning can be a preliminary to an attack, a scan for known exploits or a spreading worm and all of these are a potential threat.

Everyone, whether you are a home user or a large company, should act against the worldwide scanning, in order to reduce it. Products like *honeyd* and *LaBrea* are ready to use.

References

1. Morris, Chris. "What Do You Do After You Deploy the IDS?". January 3, 2001 URL: http://www.sans.org/newlook/resources/IDFAQ/deploy.htm
2. Staniford, Stuart. Paxson, Vern. Weaver, Nicholas. "How to 0wn the Internet in Your Spare Time" URL: http://www.icir.org/vern/papers/cdc-usenix-sec02/index.html
3. Internet Security Systems, Inc. "Executive Summary Internet Risk Summary", for March 26, 2002 through June 24 2002 URL: https://gtoc.iss.net/documents/summaryreport.pdf
4. Staniford, Stuart. Hoagland, James. McAlerney, Joseph. "Practical Automated Detection of Stealthy Portscans" URL: http://www.silicondefense.com/pptntext/Spice-JCS.pdf
5. Provos, Niels. "Honeyd - Network Rhapsody for You". URL: http://www.citi.umich.edu/u/provos/honeyd/
6. Liston, Tom. "Welcome To My Tarpit - The Tactical and Strategic Use of LaBrea" URL: http://www.hackbusters.net/LaBrea/LaBrea.txt

Additional references

7. Schlotter, Chadd. "Anti-Hacking: The Protection of Computers". April 2, 2001 URL: http://rr.sans.org/attack/antihack.php
8. Haig, Leigh. "LaBrea - A New Approach to Securing Our Networks". Match 7, 2002. URL: http://rr.sans.org/attack/labrea.php
9. Distributed Intrusion Detection System. URL: http://www.dshield.org/
10. Internet Storm Center. URL: http://isc.incidents.org/
11. Security Focus DeepSight Analyzer (ARIS). URL: http://aris.securityfocus.com/
12. HackerWatch.org (McAfee Personal Firewall). URL: http://www.hackerwatch.org/
13. Provos, Niels. "honeyd creates network schizophrenia". April 8, 2002. URL: http://archives.neohapsis.com/archives/sf/honeypots/2002-q2/0010.html
14. Homepage of the *LaBrea Tarpit.* URL: http://www.hackbusters.net/LaBrea/

Assignment 2 - Network Detects -

Detect #1 - Outbound connection attempts to a host on TCP Port 30001

Event traces

The following log is from a Tunix firewall logging to syslog. The format of the logging consists of the columns:

> Month, Day of month, hh:mm:ss, hostname (xx), process which generated the log-entry (kernel) and the log-message.

The message "redirecting TCP port 30001 to port xx (my.int.net.6:4706 -> 193.195.96.70:30001)" is best interpreted as: "Internal host my.int.net.6:portnr wants to establish a TCP connection to 193.195.96.70 on port 30001, but there is no rule for this type of connection and therefore, the connection is prohibited and redirected".

```
Jun 21 14:11:36 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1312 -> 193.195.96.70:30001)
Jun 21 14:11:36 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1312 -> 193.195.96.70:30001)
Jun 21 14:11:36 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1312 -> 193.195.96.70:30001)
Jun 21 14:11:37 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1312 -> 193.195.96.70:30001)
----- more of these lines -----
Jun 24 14:11:13 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1657 -> 193.195.96.70:30001)
Jun 24 14:11:13 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1657 -> 193.195.96.70:30001)
Jun 24 14:11:14 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1657 -> 193.195.96.70:30001)
Jun 24 14:11:14 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1657 -> 193.195.96.70:30001)
----- more of these lines -----
Jun 24 15:37:35 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1038 -> 193.195.96.70:30001)
Jun 24 15:37:36 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1038 -> 193.195.96.70:30001)
Jun 24 15:37:36 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1038 -> 193.195.96.70:30001)
Jun 24 15:37:37 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1038 -> 193.195.96.70:30001)
----- more of these lines -----
Jun 26 11:07:17 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1626 -> 193.195.96.70:30001)
Jun 26 11:07:18 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1626 -> 193.195.96.70:30001)
Jun 26 11:07:18 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1626 -> 193.195.96.70:30001)
Jun 26 11:07:19 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1626 -> 193.195.96.70:30001)
----- more of these lines -----
```

Correlated output of the tcpdump log with refused connections:

```
----- connection attempt with source port 1657 -----
14:11:13.139739 my.int.net.6.1657 > 193.195.96.70.30001: S
210723337:210723337(0) win 8192 <mss 4016> (DF)
14:11:13.139832 193.195.96.70.30001 > my.int.net.6.1657: R 0:0(0) ack
210723338 win 0

14:11:13.648646 my.int.net.6.1657 > 193.195.96.70.30001: S
210723337:210723337(0) win 8192 <mss 4016> (DF)
14:11:13.648731 193.195.96.70.30001 > my.int.net.6.1657: R 0:0(0) ack
210723338 win 0

14:11:14.195688 my.int.net.6.1657 > 193.195.96.70.30001: S
210723337:210723337(0) win 8192 <mss 4016> (DF)
14:11:14.197129 193.195.96.70.30001 > my.int.net.6.1657: R 0:0(0) ack
210723338 win 0

14:11:14.742151 my.int.net.6.1657 > 193.195.96.70.30001: S
210723337:210723337(0) win 8192 <mss 4016> (DF)
14:11:14.742237 193.195.96.70.30001 > my.int.net.6.1657: R 0:0(0) ack
210723338 win 0

----- connection attempt with source port 1038 -----
15:37:35.698708 my.int.net.6.1038 > 193.195.96.70.30001: S
261370831:261370831(0)
win 8192 <mss 4016> (DF)
15:37:35.698822 193.195.96.70.30001 > my.int.net.6.1038: R 0:0(0) ack
261370832 win 0

15:37:36.240024 my.int.net.6.1038 > 193.195.96.70.30001: S
261370831:261370831(0)
win 8192 <mss 4016> (DF)
15:37:36.240464 193.195.96.70.30001 > my.int.net.6.1038: R 0:0(0) ack 1
win 0

15:37:36.786780 my.int.net.6.1038 > 193.195.96.70.30001: S
261370831:261370831(0)
win 8192 <mss 4016> (DF)
15:37:36.786911 193.195.96.70.30001 > my.int.net.6.1038: R 0:0(0) ack 1
win 0

15:37:37.333532 my.int.net.6.1038 > 193.195.96.70.30001: S
261370831:261370831(0)
win 8192 <mss 4016> (DF)
15:37:37.333633 193.195.96.70.30001 > my.int.net.6.1038: R 0:0(0) ack 1
win 0
```

Correlated output of a tcpdump log with a connection in the honeypot-setup:

```
13:03:33.405896 my.int.net.6.4170 > 193.195.96.70.30001: S
545229220:545229220(0) win 8192 <mss 4016> (DF)
0x0000  aaaa 0300 0000 0800 4500 002c 5d97 4000       ........E..,].@.
0x0010  8006 4816 xxxx xxxx c1c3 6046 104a 7531       ..H...)...`F.Ju1
0x0020  207f 89a4 0000 0000 6002 2000 e96c 0000       ........`....l..
0x0030  0204 0fb0                                     ....
```

```
13:03:33.405976 193.195.96.70.30001 > my.int.net.6.4170: S
3032401167:3032401167(0) ack 545229221 win 9800 <mss 1960> (DF)
0x0000  aaaa 0300 0000 0800 4500 002c 4ce1 4000       ........E..,L.@.
0x0010  4006 98cc c1c3 6046 xxxx xxxx 7531 104a       @.....`F..).u1.J
0x0020  b4be c50f 207f 89a5 6012 2648 714d 0000       ........`.&HqM..
0x0030  0204 07a8                                     ....

13:03:33.406168 my.int.net.6.4170 > 193.195.96.70.30001: . ack 1 win
9800 (DF)
0x0000  aaaa 0300 0000 0800 4500 0028 5e97 4000       ........E..(^.@.
0x0010  8006 471a xxxx xxxx c1c3 6046 104a 7531       ..G...)...`F.Ju1
0x0020  207f 89a5 b4be c510 5010 2648 8afe 0000       ........P.&H....

13:03:33.406728 my.int.net.6.4170 > 193.195.96.70.30001: P 1:264(263)
ack 1 win 9800 (DF)
0x0000  aaaa 0300 0000 0800 4500 012f 5f97 4000       ........E../_.@.
0x0010  8006 4513 xxxx xxxx c1c3 6046 104a 7531       ..E...)...`F.Ju1
0x0020  207f 89a5 b4be c510 5018 2648 f769 0000       ........P.&H.i..
0x0030  4745 5420 2f69 6d61 6765 732f 3130 372e       GET./images/107.
0x0040  312f 3436 3878 3036 302e 6769 663f 6d61       1/468x060.gif?ma
0x0050  726f 6b6b 6f70 2048 5454 502f 312e 300d       rokkop.HTTP/1.0.
0x0060  0a41 6363 6570 743a 202a 2f2a 0d0a 5265       .Accept:.*/*..Re
0x0070  6665 7265 723a 2068 7474 703a 2f2f 6164       ferer:.http://ad
0x0080  732e 6d61 726f 6b6b 6f2e 6e6c 2f61 6466       s.marokko.nl/adf
0x0090  7261 6d65 2e70 6870 3f77 6861 743d 616c       rame.php?what=al
0x00a0  6c2c 5f34 3638 7836 3026 7461 7267 6574       l,_468x60&target
0x00b0  3d5f 626c 616e 6b0d 0a41 6363 6570 742d       =_blank..Accept-
0x00c0  4c61 6e67 7561 6765 3a20 6e6c 0d0a 5573       Language:.nl..Us
0x00d0  6572 2d41 6765 6e74 3a20 4d6f 7a69 6c6c       er-Agent:.Mozill
0x00e0  612f 342e 3020 2863 6f6d 7061 7469 626c       a/4.0.(compatibl
0x00f0  653b 20xx xxxx xxxx xxxx xxxx xxxx xxxx       e;.xxxxxxxxxxxxx
0x0100  xxxx xxxx xxxx xxxx xxxx xxxx xxxx 486f       xxxxxxxxxxxxxxHo
0x0110  7374 3a20 6261 6e6e 6572 7331 2e72 6573       st:.banners1.res
0x0120  756c 746f 6e6c 696e 652e 636f 6d3a 3330       ultonline.com:30
0x0130  3030 310d 0a0d 0a                             001....

13:03:33.419257 193.195.96.70.30001 > my.int.net.6.4170: P 1:40(39) ack
264 win 9800 (DF)
13:03:33.540527 my.int.net.6.4170 > 193.195.96.70.30001: . ack 40 win
9761 (DF)
13:03:33.554171 193.195.96.70.30001 > my.int.net.6.4170: P 40:222(182)
ack 264 win 9800 (DF)
13:03:33.759185 my.int.net.6.4170 > 193.195.96.70.30001: . ack 222 win
9579 (DF)

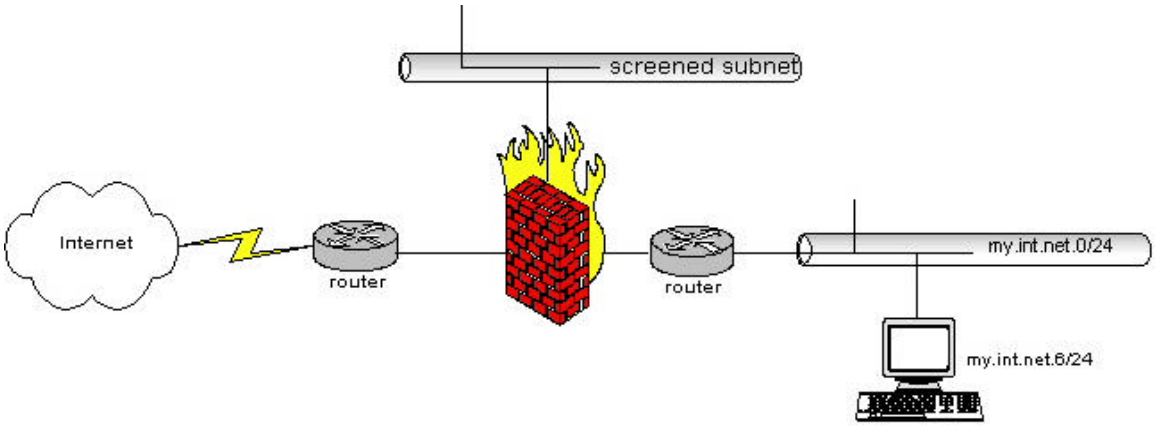
13:05:34.087903 my.int.net.6.4170 > 193.195.96.70.30001: F 264:264(0)
ack 222 win 9579 (DF)
13:05:34.089730 193.195.96.70.30001 > my.int.net.6.4170: F 222:222(0)
ack 265 win 9800 (DF)
13:05:34.089950 my.int.net.6.4170 > 193.195.96.70.30001: . ack 223 win
9579 (DF)
```

1. Source of Trace

The source of this trace is obtained from a Tunix firewall on the network of a customer which is being monitored by my company. This firewall is a multihomed system,

connected to the Internet, the internal network and a screened subnet. The following diagram is only a simplified representation to illustrate the basic setup and does not necessarily represent the real setup which is far more complex.

Simplified network diagram:



2. Detect was generated by

The detect was made by examine the log files of the customer's firewall. The connection attempts were blocked by the firewall and reported. There were over 3000 connection attempts from this internal source-IP to the same Internet host on TCP port 30001 logged for a couple of days in a row. It was possible to extend this logging by generating a tcpdump log of the blocked connections and a tcpdump log of a "successful" connection using a simple honeypot setup. This scenario was possible because these connection-attempts were successfully blocked by the firewall and the environment was under control.

The user of this specific source-IP was interviewed by the local system administrator, but there was no indication about how this traffic was generated.

The fact that one internal host tries to connect to a specific Internet host on a high port (TCP 30001), for several days at various intervals, is suspicious. TCP port 30001 is a default port used by the Trojan "ErrOr32"[19+20]

3. Probability the source address was spoofed

The TCP-SYN package with an internal source-IP is answered with a TCP-RST package. There are 3 immediate retries of the same TCP-SYN package, within 1 second, which *could* indicate that the TCP-RST packets arrive at the source-IP, or that the originator is sending 4 forged TCP-SYN packets in a row without waiting for a response. There is no evidence that the TCP-RST packets arrive at the originator of the connection. This evidence could be acquired by monitoring the internal router and the MAC-addresses, but this should be done in conjunction with the local system administrator which was not possible at this time.

There was no evidence found in the firewall logs that packages with an internal source address from outside the internal network passed through the firewall, so the TCP-SYN package is most probably originating from the internal network.

After setting up the honeypot, there is evidence of a completed three-way TCP handshake, TCP session and four-way TCP-FIN sequence, indicating that the originating host is indeed on the internal network and that the address is not spoofed. The traffic stopped from occurring right after changing the specific host to use the proxy.

4. Description of attack

This is standard HTTP-protocol traffic connecting to a web-server listening on port 30001. The web server listening seems to be a **Boa/0.94.12rc8** server and the welcome page indicates that the host-name is *banners1.resultonline.com*.
This is the GET-request on port 30001 with the server-response:

GET /index.html HTTP/1.0

HTTP/1.0 200 OK
Date: Sun, DD Jun 2002 12:51:31 GMT
Server: **Boa/0.94.12rc8**
Connection: close
Content-Length: 5839
Last-Modified: Mon, 01 Oct 2001 15:49:12 GMT
Content-Type: text/html

The DNS-administration seems to be inconsistent for *banners.resultonline.com* and should be updated. There are also two different geographical locations for the resultonline.com webservers:

Authority records for *banners1.online.com*:

dig **banners1.resultonline.com**
;; ANSWERS:
banners1.resultonline.com. 9357 A **193.195.96.70**
;; AUTHORITY RECORDS:
resultonline.com. 79003 NS bram.wiwo.nl.
resultonline.com. 79003 NS sec-ns.wiwo.nl.
resultonline.com. 79003 NS taz.wiwo.nl.

Pointer-record for *193.195.96.*70:

dig -x **193.195.96.70**
;; ANSWERS:
70.96.195.193.in-addr.arpa. 11453 PTR **banners.resultonline.com**.
;; AUTHORITY RECORDS:
96.195.193.in-addr.arpa. 251835 NS ns1.demon.co.uk.
96.195.193.in-addr.arpa. 251835 NS ns2.demon.net.
96.195.193.in-addr.arpa. 251835 NS ns0.demon.co.uk.

Authority records for *banners.online.com*:

dig **banners.resultonline.com**
;; ANSWERS:
banners.resultonline.com. 9462 A **146.101.66.152**
;; AUTHORITY RECORDS:
resultonline.com. 79035 NS bram.wiwo.nl.
resultonline.com. 79035 NS sec-ns.wiwo.nl.
resultonline.com. 79035 NS taz.wiwo.nl.

Pointer-record for *146.101.66.152*:

dig -x **146.101.66.152**
;; ANSWERS:
152.66.101.146.in-addr.arpa. 16790 PTR **nl-wit-01.amst2.eu.psigh.com**.
;; AUTHORITY RECORDS:
66.101.146.in-addr.arpa. 35520 NS pri3.dns.psinet.fr.
66.101.146.in-addr.arpa. 35520 NS pri1.dns.psinet.fr.
66.101.146.in-addr.arpa. 35520 NS pri2.dns.psinet.fr.

5. Attack mechanism

Traffic is redirected to the *banners1.resultonline.com* server by HTML-links who are dynamically generated on various webpages. By analyzing the traffic preceding this connection, it was possible to reveal the webpage which caused the connection request for *banners1.resultonline.com* on port 30001. The URL: http://ajax.network.to (and the related URLs http://feyenoord.network.to and http://psv.network.to) have a randomized banner in the top of their homepage. One of these random banners is redirecting to *banners1.resultonline.com:30001*.

6. Correlations

At first sight, it looked like the internal host was using the client version of the Trojan *Err0r32* program. There are no specific CVE-numbers, but the URLs:

- http://www.dark-e.com/archive/trojans/err32/beta/
- http://www.megasecurity.org/trojans/e/error32/Error32.html

explain the Trojan behavior as a server and client application. It is possible to gain remote control over an infected host with this Trojan.

After looking closer into the logs of many other firewalls that are being monitored by my company, it turned out that a few customer's experienced the same connection attempts to the same destination IP, so the possibility that the *Err0r32-client* was used became less obvious. In all cases in turned out to be the same benign traffic.

There was no specific information found according the targeted webserver.

7. Evidence of active targeting

There is no evidence of active targeting. The HTTP-protocol is used to display random banners in a browser application when visiting a web-page which happens to have a link to a web-server listening on port 30001, which is unusual for a webserver.

8. Severity

severity = (criticality + lethality) - (system countermeasures + network countermeasures)
Each element is ranked in a scale from 1-5, with 1 being low and 5 being high.

Criticality: **3**
The targeted system is an Internet host (webserver) but the traffic originated from the internal host was benign.
Lethality: **1**
The traffic causes no damage.
System countermeasures: **3**
The internal host runs updated anti-virus software and has recently patches installed. There is nothing known about the targeted web-server. There is a possibility that the user could have installed the Trojan client.
Network countermeasures: **4**
The connection attempts were blocked by a firewall.

Severity = (3 + 1) - (3 + 4) = -3

9. Defensive recommendation

Defenses are fine since the attempted connection was blocked by the firewall and the internal host happens to run updated anti-virus software. However by using the application level proxy which resides on the firewall, protocol screening would be applied **and** the application level proxy could be allowed to setup the connection, showing the web-page as intended by the developer. The local administrator was informed and appropriate actions were taken.

10. Multiple choice test question

This log has been generated by a firewall's syslog which happens to be the default route of the internal host *my.int.net.6.* It reports on attempted connections to host 193.195.96.70 on port 30001, which are being blocked.

```
Jun 26 11:07:17 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1626 -> 193.195.96.70:30001)
Jun 26 11:07:18 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1626 -> 193.195.96.70:30001)
Jun 26 11:07:18 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1626 -> 193.195.96.70:30001)
```

```
Jun 26 11:07:19 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:1626 -> 193.195.96.70:30001)
----- more of these lines -----
Jun 26 17:28:25 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:2066 -> 193.195.96.70:30001)
Jun 26 17:28:25 xx kernel: redirecting TCP port 30001 to port xx
(my.int.net.6:2066 -> 193.195.96.70:30001)
```

Knowing that *my.int.net.6* is an IP-address on the internal network protected by a firewall, which of the following is most likely shown in the trace above?

a) This internal host is infected by a Trojan
b) This internal host tries to connect to an Internet host on a high port.
c) This is a spoofed packet, generated from within the internal network.
d) This is a spoofed packet, generated from outside the internal network.

Answer: 'b'
Although it looks like Trojan behavior, you can't definite tell from the trace it selves.

Detect #2 - Loadbalancing versus reconnaissance attack.

Event traces

The 4 event traces below are sanitized in a way to protect the guilty and the innocent. The ip-ranges aa.bbb.ccc.128/25 and dd.eee.ff.128/25 are two ip-ranges registered to the same company, connected to two different ISPs.

Trace 1 - Excerpt from a Snort Alert log-file:

```
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
08/08-15:32:17.238726 aa.bbb.ccc.130:80 -> my.net.two.126:53
TCP TTL:46 TOS:0x0 ID:36803 IpLen:20 DgmLen:40
***A**** Seq: 0x22A  Ack: 0x0  Win: 0x578  TcpLen: 20

[**] [1:504:2] MISC source port 53 to <1024 [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/08-15:32:17.238983 aa.bbb.ccc.130:53 -> my.net.two.126:53
TCP TTL:46 TOS:0x0 ID:36804 IpLen:20 DgmLen:40
******S* Seq: 0xDD301D49  Ack: 0x0  Win: 0x578  TcpLen: 20

[**] [1:654:5] SMTP RCPT TO overflow [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
08/08-15:32:23.645566 aa.bbb.ccc.190:12736 -> my.net.two.126:25
TCP TTL:240 TOS:0x10 ID:0 IpLen:20 DgmLen:1821
***AP*** Seq: 0xB4073D13  Ack: 0xDB0A645B  Win: 0x2238  TcpLen: 20
```

```
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
08/08-16:19:51.250467 aa.bbb.ccc.130:80 -> my.net.one.39:53
TCP TTL:46 TOS:0x0 ID:50271 IpLen:20 DgmLen:40
***A**** Seq: 0x1AC  Ack: 0x0  Win: 0x578  TcpLen: 20

[**] [1:504:2] MISC source port 53 to <1024 [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/08-16:19:51.251073 aa.bbb.ccc.130:53 -> my.net.one.39:53
TCP TTL:46 TOS:0x0 ID:50272 IpLen:20 DgmLen:40
******S* Seq: 0x7B52E6F  Ack: 0x0  Win: 0x578  TcpLen: 20

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
08/09-18:32:07.901594 aa.bbb.ccc.130:80 -> my.net.two.126:25
TCP TTL:46 TOS:0x0 ID:40087 IpLen:20 DgmLen:40
***A**** Seq: 0x3C4  Ack: 0x0  Win: 0x578  TcpLen: 20

[**] [1:654:5] SMTP RCPT TO overflow [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
08/09-18:32:14.238734 aa.bbb.ccc.195:5878 -> my.net.two.126:25
TCP TTL:240 TOS:0x10 ID:0 IpLen:20 DgmLen:3294
***AP*** Seq: 0x5E88DF60  Ack: 0xFFDB7A6C  Win: 0x2238  TcpLen: 20

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
08/09-18:35:04.206237 aa.bbb.ccc.130:80 -> my.net.one.39:53
TCP TTL:46 TOS:0x0 ID:40994 IpLen:20 DgmLen:40
***A**** Seq: 0xDE  Ack: 0x0  Win: 0x578  TcpLen: 20

[**] [1:504:2] MISC source port 53 to <1024 [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/09-18:35:04.206817 aa.bbb.ccc.130:53 -> my.net.one.39:53
TCP TTL:46 TOS:0x0 ID:40995 IpLen:20 DgmLen:40
******S* Seq: 0x87C9F351  Ack: 0x0  Win: 0x578  TcpLen: 20
```

The following two logs are from firewalls logging to syslog. The format of the logging consists of the columns:

> Month, Day of month, hh:mm:ss, hostname (host1 or host2), process which generated the log-entry (kernel, mpgr, txp, smtp), and the log-message.

The message "redirecting UDP port 37852 to port xx (aa.bbb.ccc.130:55 -> my.net.one:37852)" is best interpreted as: "The UDP packet 37852 coming from aa.bb.ccc.130 to my.net.one.39 is prohibited and discarded". The other log-messages are explained between '--<*explanation*>--' as they occur in the logging.

The time setting between the alert logging and the two firewall logs are not synchronized. The time-gap between the NIDS and host1 is about 18 seconds and can be correlated through entries in the syslog of host 1 (trace 2) which match the entries in the tcpdump log (trace 4) which was made on the NIDS. The time-gap between the NIDS and host2 is about 23 seconds and can be correlated through entries in the alert log (trace 1) with matching connections in the syslog of host2 (trace 3).

Trace 2 - Syslog for my.net.one (host1):
The highlighted text is to synchronize the time with the tcpdump log (trace 4) for my.net.one.

```
Aug  8 16:19:34 host1 kernel: redirecting UDP port 37852 to port xx
(aa.bbb.ccc.130:55 -> my.net.one.39:37852)
Aug  8 16:19:34 host1 kernel: redirecting UDP port 37852 to port xx
(dd.eee.ff.130:53 -> my.net.one.39:37852)

Aug  9 18:34:46 host1 kernel: redirecting UDP port 37852 to port xx
(aa.bbb.ccc.130:55 -> my.net.one.39:37852)
Aug  9 18:34:46 host1 kernel: redirecting UDP port 37852 to port xx
(dd.eee.ff.130:53 -> my.net.one.39:37852)

The same type of traffic also occurred in July. Two UDP packets, 1 from
aa.bbb.ccc.130:55 and 1 from dd.eee.ff.130:53. The date and time of
these packets:
                  Jul  1 12:10:23              Jul 22 18:44:06
                  Jul  1 16:51:58              Jul 23 08:32:22
```

Trace 3 - Syslog for my.net.two (host2):

```
--<Normal mail exchange from my.dom to other.dom>--
Aug  7 15:06:09 host2 mprg[18122]: xxxxx: to=<o.ailer1@other.dom>,
ctladdr=<m.ailer1@my.dom> (0/0), delay=00:00:03, xdelay=00:00:02,
mailer=smtp, relay=mail1.other.dom. [aa.bbb.ccc.195], stat=Sent
(smtp1.other.dom: Message accepted for delivery)

--<After the mail exchange is closed, 2 UDP 37852 packets to my.dom>--
Aug  7 15:06:15 host2 kernel: redirecting UDP port 37852 to port xx
(aa.bbb.ccc.130:55 -> my.net.two.126:37852)
Aug  7 15:06:15 host2 kernel: redirecting UDP port 37852 to port xx
(dd.eee.ff.130:53 -> my.net.two.126:37852)

--<and mail to confirm that the previous message was recieved>--
Aug  7 15:06:15 host2 txp[17882]: exec aa.bbb.ccc.195:9830 to
my.net.two.126:25 ( )
Aug  7 15:06:17 host2 smtp[17882]: host=unknown/aa.bbb.ccc.195
bytes=2081 from=<> to=<m.ailer1@my.dom>
Aug  7 15:06:17 host2 smtp[17882]: exiting host=unknown/aa.bbb.ccc.195
bytes=2081

--<Two UDP 37852 packets from two different IP-numbers>--
Aug  8 15:32:00 host2 kernel: redirecting UDP port 37852 to port xx
(aa.bbb.ccc.130:55 -> my.net.two.126:37852)
Aug  8 15:32:00 host2 kernel: redirecting UDP port 37852 to port xx
(dd.eee.ff.130:53 -> my.net.two.126:37852)

--<followed by a mail, sent by other.dom to my.dom>--
Aug  8 15:32:01 host2 txp[5923]: exec aa.bbb.ccc.190:12736 to
my.net.two.126:25 ( )
Aug  8 15:32:07 host2 smtp[5923]: host=unknown/aa.bbb.ccc.190 bytes=1915
from=<o.ailer1@other.dom> to=<m.ailer1@my.dom>
Aug  8 15:32:07 host2 smtp[5923]: exiting host=unknown/aa.bbb.ccc.190
bytes=1915
```

```
--<Normal mail exchange from my.dom to other.dom>--
Aug  8 16:19:35 host2 mprg[29371]: xxxxx: to=<o.ailer1@other.dom>,
ctladdr=<m.ailer1@my.dom> (0/0), delay=00:00:02, xdelay=00:00:01,
mailer=smtp, relay=ns1.other.dom. [dd.eee.ff.194], stat=Sent
(smtp1.other.dom: Message accepted for delivery)

--<Normal mail exchange from other.dom to my.dom>--
Aug  8 16:19:55 host2 txp[29035]: exec dd.eee.ff.194:4253 to
my.net.two.126:25 ( )
Aug  8 16:19:56 host2 smtp[29035]: connect
host=ns1.other.dom/dd.eee.ff.194
Aug  8 16:19:56 host2 smtp[29035]: host=ns1.other.dom/dd.eee.ff.194
bytes=2099 from=<> to=<m.ailer1@my.dom>
Aug  8 16:19:56 host2 smtp[29035]: exiting
host=ns1.other.dom/dd.eee.ff.194 bytes=2099

--<Two UDP 37852 packets from two different IP-numbers>--
Aug  9 18:31:50 host2 kernel: redirecting UDP port 37852 to port xx
(aa.bbb.ccc.130:5880 -> my.net.two.126:37852)
Aug  9 18:31:50 host2 kernel: redirecting UDP port 37852 to port xx
(dd.eee.ff.130:5878 -> my.net.two.126:37852)

--<followed by mail, sent by other.dom to my.dom>--
Aug  9 18:31:50 host2 txp[28458]: exec aa.bbb.ccc.195:5878 to
my.net.two.126:25 ( )Aug  9 18:31:55 ns smtp[28458]: connect
host=host3.other.dom/aa.bbb.ccc.195
Aug  9 18:33:28 host2 smtp[28458]: host=host3.other.dom/aa.bbb.ccc.195
bytes=3992922 from=<o.ailer2@other.dom> to=<m.ailer2@my.dom>
Aug  9 18:33:28 host2 smtp[28458]: exiting
host=host3.other.dom/aa.bbb.ccc.195 bytes=3992922
```

Trace 4 - Log from tcpdump (-v option) , running incidentally on the NIDS for several selected TCP-ports for the my.net.one address-space. This dump is useful to synchronize with the alert log and to determine the overall traffic flow.

Part 1

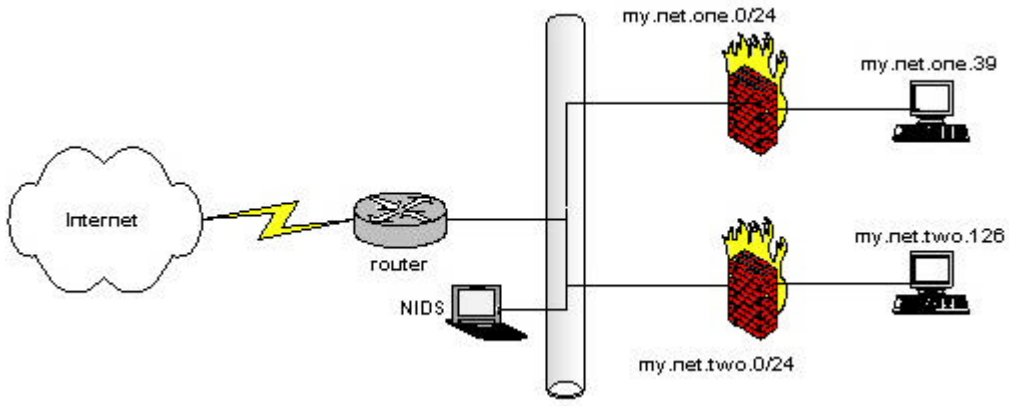
```
16:19:51.243468 dd.eee.ff.194.53 > my.net.one.39.53:  [udp sum ok] 20870
A? host.my.dom. [|domain] (ttl 48, id 23462, len 61)
16:19:51.250470 aa.bbb.ccc.130.80 > my.net.one.39.53: . [tcp sum ok] ack
0 win 1400 (ttl 46, id 50271, len 40)
16:19:51.251075 aa.bbb.ccc.130.53 > my.net.one.39.53: S [tcp sum ok]
129314415:129314415(0) win 1400 (ttl 46, id 50272, len 40)
16:19:51.256206 my.net.one.39.53 > dd.eee.ff.194.53:  [udp sum ok]
20870* 1/3/3 host.my.dom. A xxx.yy.zzz.3 (167) (ttl 62, id 46313, len
195)
16:19:51.261849 my.net.one.39.53 > aa.bbb.ccc.130.80: R [tcp sum ok]
0:0(0) win 0 (ttl 62, id 46316, len 40)
16:19:51.262670 my.net.one.39.53 > aa.bbb.ccc.130.53: S [tcp sum ok]
1205388193:1205388193(0) ack 129314416 win 8192 <mss 1460> (DF) (ttl 62,
id 46317, len 44)
16:19:51.379821 dd.eee.ff.130.53 > my.net.one.39.37852:  [udp sum ok] 0
[0q] (10) (ttl 49, id 50275, len 38)
16:19:51.388672 aa.bbb.ccc.130.53 > my.net.one.39.53: R [tcp sum ok]
129314416:129314416(0) win 1400 (ttl 46, id 50274, len 40)
16:19:51.392761 aa.bbb.ccc.130.53 > my.net.one.39.53: R [tcp sum ok]
129314416:129314416(0) win 1400 (ttl 49, id 50276, len 40)
```

```
18:35:04.195780 dd.eee.ff.189.16093 > my.net.one.39.53:  [udp sum ok]
2018 A? host.my.dom. [|domain] (ttl 48, id 27864, len 61)
18:35:04.206236 aa.bbb.ccc.130.80 > my.net.one.39.53: . [tcp sum ok] ack
0 win 1400 (ttl 46, id 40994, len 40)
18:35:04.206816 aa.bbb.ccc.130.53 > my.net.one.39.53: S [tcp sum ok]
2278159185:2278159185(0) win 1400 (ttl 46, id 40995, len 40)
18:35:04.208753 my.net.one.39.53 > dd.eee.ff.189.16093:  [udp sum ok]
2018* 1/3/3 host.my.dom. A xxx.yy.zzz.3 (167) (ttl 62, id 29417, len
195)
18:35:04.217833 my.net.one.39.53 > aa.bbb.ccc.130.80: R [tcp sum ok]
0:0(0) win 0 (ttl 62, id 29420, len 40)
18:35:04.218653 my.net.one.39.53 > aa.bbb.ccc.130.53: S [tcp sum ok]
3895922867:3895922867(0) ack 2278159186 win 8192 <mss 1460> (DF) (ttl
62, id 29421, len 44)
18:35:04.339561 dd.eee.ff.130.53 > my.net.one.39.37852:  [udp sum ok] 0
[0q] (10) (ttl 49, id 41015, len 38)
18:35:04.348440 aa.bbb.ccc.130.53 > my.net.one.39.53: R [tcp sum ok]
2278159186:2278159186(0) win 1400 (ttl 46, id 41014, len 40)
```

1. Source of Trace

The source of this trace is a combined network running 2 different C-address blocks which is monitored by my company. The NIDS and the two Tunix firewalls which generated the syslog are not on the same logical network.

Simplified network diagram:



2. Detect was generated by

Snort NIDS version 1.8.6 and two different Tunix firewalls logging to syslog. The Snort Alerts were the stimulus to start analyzing this detect. The two firewall logs and the tcpdump log provided more insight in the alerts.

3. Probability the source address was spoofed

The UDP packets with destination port 37258 and the DNS requests could easily be spoofed. The same is true for the ACK's without preceding SYN-ACK, however there

was also a completed 3-way TCP handshake which is difficult to spoof. The TTL-values in the tcpdump log are the same for all packets originated from the same IP-address, this makes spoofing unlikely. There is however one contradiction. Host aa.bbb.ccc.130 arrives with a TTL value of 46, except one TCP-RST packet which arrives with a TTL value of 49. This TTL value happens to be the same as the preceding UDP packet with destination port 37852.

```
16:19:51.379821 dd.eee.ff.130.53 > my.net.one.39.37852:  [udp sum ok] 0
[0q] (10) (ttl 49, id 50275, len 38)
16:19:51.388672 aa.bbb.ccc.130.53 > my.net.one.39.53: R [tcp sum ok]
129314416:129314416(0) win 1400 (ttl 46, id 50274, len 40)
16:19:51.392761 aa.bbb.ccc.130.53 > my.net.one.39.53: R [tcp sum ok]
129314416:129314416(0) win 1400 (ttl 49, id 50276, len 40)
```

This could be explained by the fact that IP ‘dd.eee.ff.130’ and ‘aa.bbb.ccc.130’ reside on the same host or that a NATting is in the traversal path.

The syslog for my.net.two (trace 3) shows two successful mail exchange’s, preceded by the UDP 37852 packets. (**aa.bbb.ccc.190:12736** and **aa.bbb.ccc.195:5878**) which makes the probability of spoofing low.

4. Description of attack

This is probably some kind of load-balancing device to determine the most efficient connection to use. It looks like *LinkProof*, a product made by Radware, http://www.radware.com/content/products/link.asp. Radware claims that LinkProof the fastest content delivery for networks with multiple connections to the Internet ensures.
There is no CVE number for *LinkProof.*
The two ranges of the source-IP numbers belong to the same organization, but different ISP’s, what could confirm the load balancing theory.

5. Attack mechanism

There are no evidences found of a netscan or hostscan done by the source-IP ranges. All traffic is targeted at a specific host. By examining the first part of the tcpdump log and the syslog of my.net.one.host1, the following connect pattern is discovered:

- Connection 1
  dd.eee.ff.194:53 → my.net.one.39:53 = => DNS-query using UDP port 53
  This query is being answered since my.net.one:53 is a public DNS-server and the request is legitimate.
- Connection 2
  aa.bbb.ccc.130:80 → my.net.one.39:53 = => TCP-ACK(0) packet.
  There is no preceding SYN-ACK therefore my.net.one.39:53 sends back a TCP-RST to tear down the connection.

- Connection 3
  aa.bbb.ccc.130:53 → my.net.one.39:53 = => TCP-SYN
  This TCP-SYN is being answered by my.net.one.39:53 with a SYN-ACK but aa.bbb.ccc.130:53 does not complete the three-way handshake, but sends a TCP-RST to tear down the connection. In this specific case, aa.bbb.ccc.130:53 sends two TCP-RST packets with different TTL's (49 and 46) to tear down the connection. This could indicate that there is some kind of duplicating device.
- Connection 4
  aa.bbb.ccc.130:55 → my.net.one.39:37852 = => UDP (0) packet.
  This packet could be a probe for a specific service or it's purpose is to gather information by the ICMP package which could be returned by the host (i.e. A port-unreachable message)
- Connection 5
  dd.eee.ff.130:53 → my.net.one:37852 = => UDP (0) packet.
  This packet has a different source address, but could be from the same host (aa.bbb.ccc.130) with a different connection and traverse path. The TTL of this packet is 49 which is the same as the second TCP-RST packet in connection 3. This could indicate that there is some kind of network address translation which did not do the job very well.

The second part of the tcpdump-log (trace 3) shows the same pattern, except that the host in connection 1 is different and uses a high source-port for the DNS-query which could indicate that the attacker.net use two different DNS-resolver implementations at the same subnet (both give a TTL of 48). The second difference is that there is only one TCP-RST packet to tear down connection 3.

It is very likely that attacker.net uses some device to detect the most efficient connection by sending unsolicited TCP-ACK and UDP packet to a port which is very likely not used by a service, to elicit a TCP-RST or ICMP "port unreachable" message from which they can determine the fastest connection.

Trace 3 shows that whenever a connection is initiated by a host on attacker.net, this connection is preceded by the UDP 37852 packet (and trace 4 shows that this UDP 37852 packet is preceded by other connections). They could also use ICMP echo-requests to get responses, but most perimeter devices are configured not to let ICMP traffic through, which makes them useless. The traces 1-4 do not mention ICMP traffic because they were configured to ignore ICMP packets so that does not implicate that there are no ICMP-requests.

There is evidence in trace 3 that the traffic which generates the alerts is followed by benign traffic (mail-exchange), so the load-balancing theory could be true.

6. Correlations

John Benninghoff reported a similar detect and an explanation from the owner of the offending IP-address at http://www.sans.org/y2k/031401.htm This attack used in also

ICMP echo-request packets and never used a source port 55 for the second UDP packet targeted at port 37852. He also mentioned RadWare but the whitepapers at http://www.radware.com/content/products/link.asp do not contain the technical information that is needed to confirm the traffic flow.
At http://www.telecomdatacom.net/radware-linkproof-whitepaper.asp, I found a more detailed overview of the features.

Another report at http://www.incidents.org/archives/y2k/032401-1230.htm is from Gary Portnoy which also reports source port 55 and a second source-IP. Even more matching correlations can be found in the thread which can be found at http://www.incidents.org/archives/intrusions/msg08119.html and the follow-up at http://www.incidents.org/archives/intrusions/msg08129.html.

Dshield reports (http://www.dshield.org/ipinfo.php?ip=aa.bbb.ccc.130) this source-IP, but there was (at this time) no Fightback sent.

7. Evidence of active targeting

There is no evidence of active targeting at a specific host. The traffic which generates the alerts are in conjunction with normal traffic exchange between the source and destination networks.

8. Severity

severity = (criticality + lethality) - (system countermeasures + network countermeasures)
Each element is ranked in a scale from 1-5, with 1 being low and 5 being high.

Criticality: **4**
The targeted systems are DNS-servers so criticality is high, but the traffic was mainly used for reconnaissance.
Lethality: **2**
The traffic causes no damage, but the DNS-servers have to deal with the connection requests.
System countermeasures: **3**
The DNS servers are maintained at regular intervals and patch levels are current.
Network countermeasures: **4**
The connection attempts which are not allowed, were successfully blocked by firewalling devices.

Severity = (4 + 2) - (3 + 4) = -1

9. Defensive recommendation

Defenses are fine since the attack was blocked by firewalls. As an extra confirmation, I asked for more information by E-mail at the registrar of the 2 IP-ranges. This E-mail has

not been answered. This kind of traffic could be an advantage to the hackers community, to perform another kind of stealth scanning.

10. Multiple choice test question

```
16:19:51.243468 dd.eee.ff.194.53 > my.net.one.39.53:  [udp sum ok] 20870
A? host.my.dom. [|domain] (ttl 48, id 23462, len 61)

16:19:51.250470 aa.bbb.ccc.130.80 > my.net.one.39.53: . [tcp sum ok] ack
0 win 1400 (ttl 46, id 50271, len 40)

16:19:51.251075 aa.bbb.ccc.130.53 > my.net.one.39.53: S [tcp sum ok]
129314415:129314415(0) win 1400 (ttl 46, id 50272, len 40)

16:19:51.256206 my.net.one.39.53 > dd.eee.ff.194.53:  [udp sum ok]
20870* 1/3/3 host.my.dom. A xxx.yy.zzz.3 (167) (ttl 62, id 46313, len
195)

16:19:51.261849 my.net.one.39.53 > aa.bbb.ccc.130.80: R [tcp sum ok]
0:0(0) win 0 (ttl 62, id 46316, len 40)

16:19:51.262670 my.net.one.39.53 > aa.bbb.ccc.130.53: S [tcp sum ok]
1205388193:1205388193(0) ack 129314416 win 8192 <mss 1460> (DF) (ttl 62,
id 46317, len 44)

16:19:51.379821 dd.eee.ff.130.53 > my.net.one.39.37852:  [udp sum ok] 0
[0q] (10) (ttl 49, id 50275, len 38)

16:19:51.388672 aa.bbb.ccc.130.53 > my.net.one.39.53: R [tcp sum ok]
129314416:129314416(0) win 1400 (ttl 46, id 50274, len 40)

16:19:51.392761 aa.bbb.ccc.130.53 > my.net.one.39.53: R [tcp sum ok]
129314416:129314416(0) win 1400 (ttl 49, id 50276, len 40)
```

Which of the following is most likely shown in the tcpdump log above?

a) Host aa.bbb.ccc.130 is a secondary dns server for host my.net.one.39.
b) Host aa.bbb.ccc.130 runs a webserver and a dns server.
c) This is proof of a poor connection between two networks.
d) This is some kind of reconnaissance traffic, done by host aaa.bbb.ccc.130.

Answer: ‘d’
Note the UDP 37852, the unsolicited TCP-ACK and the TCP-RST packets.

Detect #3 - SNMP "public" access attempt

Event traces

Trace 1 - Snort log with payload included.

```
08/12-02:21:55.298255 216.236.156.62:29158 -> my.own.net.0:161
UDP TTL:106 TOS:0x0 ID:25600 IpLen:20 DgmLen:265 Len: 245
30 81 EA 02 01 00 04 06 70 75 62 6C 69 63 A1 81  0.......public..
DC 02 01 00 02 01 00 02 01 00 30 81 D0 30 0B 06  ..........0..0..
07 2B 06 01 02 01 01 01 05 00 30 0B 06 07 2B 06  .+........0...+.
01 02 01 01 03 05 00 30 0B 06 07 2B 06 01 02 01  .......0...+....
01 05 05 00 30 0D 06 09 2B 06 01 02 01 02 02 01  ....0...+.......
06 05 00 30 0D 06 09 2B 06 01 02 01 04 14 01 01  ...0...+........
05 00 30 0E 06 0A 2B 06 01 02 01 19 03 02 01 03  ..0...+.........
05 00 30 10 06 0C 2B 06 01 04 01 0B 02 03 09 01  ..0...+.........
01 07 05 00 30 10 06 0C 2B 06 01 04 01 0B 02 03  ....0...+.......
09 05 01 03 05 00 30 10 06 0C 2B 06 01 04 01 0B  ......0...+.....
02 04 03 08 03 02 05 00 30 10 06 0C 2B 06 01 04  ........0...+...
01 0B 02 04 03 08 03 03 05 00 30 0F 06 0B 2B 06  ..........0...+.
01 04 01 0B 02 04 03 0A 07 05 00 30 0F 06 0B 2B  ...........0...+
06 01 04 01 0B 02 04 03 0A 0D 05 00 30 0F 06 0B  ............0...
2B 06 01 04 01 0B 02 04 03 0D 01 05 00           +............
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
08/14-08:33:58.481328 12.155.155.17:50343 -> my.own.net.254:161
UDP TTL:109 TOS:0x0 ID:35213 IpLen:20 DgmLen:265 Len: 245
30 81 EA 02 01 00 04 06 70 75 62 6C 69 63 A1 81  0.......public..
DC 02 01 00 02 01 00 02 01 00 30 81 D0 30 0B 06  ..........0..0..
07 2B 06 01 02 01 01 01 05 00 30 0B 06 07 2B 06  .+........0...+.
01 02 01 01 03 05 00 30 0B 06 07 2B 06 01 02 01  .......0...+....
01 05 05 00 30 0D 06 09 2B 06 01 02 01 02 02 01  ....0...+.......
06 05 00 30 0D 06 09 2B 06 01 02 01 04 14 01 01  ...0...+........
05 00 30 0E 06 0A 2B 06 01 02 01 19 03 02 01 03  ..0...+.........
05 00 30 10 06 0C 2B 06 01 04 01 0B 02 03 09 01  ..0...+.........
01 07 05 00 30 10 06 0C 2B 06 01 04 01 0B 02 03  ....0...+.......
09 05 01 03 05 00 30 10 06 0C 2B 06 01 04 01 0B  ......0...+.....
02 04 03 08 03 02 05 00 30 10 06 0C 2B 06 01 04  ........0...+...
01 0B 02 04 03 08 03 03 05 00 30 0F 06 0B 2B 06  ..........0...+.
01 04 01 0B 02 04 03 0A 07 05 00 30 0F 06 0B 2B  ...........0...+
06 01 04 01 0B 02 04 03 0A 0D 05 00 30 0F 06 0B  ............0...
2B 06 01 04 01 0B 02 04 03 0D 01 05 00           +............
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
08/15-02:47:09.174882 216.236.156.62:48249 -> my.own.net.1:161
UDP TTL:106 TOS:0x0 ID:63089 IpLen:20 DgmLen:265 Len: 245
30 81 EA 02 01 00 04 06 70 75 62 6C 69 63 A1 81  0.......public..
DC 02 01 00 02 01 00 02 01 00 30 81 D0 30 0B 06  ..........0..0..
07 2B 06 01 02 01 01 01 05 00 30 0B 06 07 2B 06  .+........0...+.
01 02 01 01 03 05 00 30 0B 06 07 2B 06 01 02 01  .......0...+....
01 05 05 00 30 0D 06 09 2B 06 01 02 01 02 02 01  ....0...+.......
06 05 00 30 0D 06 09 2B 06 01 02 01 04 14 01 01  ...0...+........
05 00 30 0E 06 0A 2B 06 01 02 01 19 03 02 01 03  ..0...+.........
05 00 30 10 06 0C 2B 06 01 04 01 0B 02 03 09 01  ..0...+.........
01 07 05 00 30 10 06 0C 2B 06 01 04 01 0B 02 03  ....0...+.......
09 05 01 03 05 00 30 10 06 0C 2B 06 01 04 01 0B  ......0...+.....
02 04 03 08 03 02 05 00 30 10 06 0C 2B 06 01 04  ........0...+...
01 0B 02 04 03 08 03 03 05 00 30 0F 06 0B 2B 06  ..........0...+.
01 04 01 0B 02 04 03 0A 07 05 00 30 0F 06 0B 2B  ...........0...+
06 01 04 01 0B 02 04 03 0A 0D 05 00 30 0F 06 0B  ............0...
2B 06 01 04 01 0B 02 04 03 0D 01 05 00           +............
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Trace 2 - Tcpdumplog (with -v option) of the same packets.

```
02:21:55.298255 216.236.156.62.29158 > my.own.net.0.161:  [udp sum ok] {
SNMPv1 { GetNextRequest(220) R=0  .1.3.6.1.2.1.1.1 .1.3.6.1.2.1.1.3
.1.3.6.1.2.1.1.5 .1.3.6.1.2.1.2.2.1.6 .1.3.6.1.2.1.4.20.1.1
.1.3.6.1.2.1.25.3.2.1.3 .1.3.6.1.4.1.11.2.3.9.1.1.7
.1.3.6.1.4.1.11.2.3.9.5.1.3 .1.3.6.1.4.1.11.2.4.3.8.3.2
.1.3.6.1.4.1.11.2.4.3.8.3.3 .1.3.6.1.4.1.11.2.4.3.10.7
.1.3.6.1.4.1.11.2.4.3.10.13 .1.3.6.1.4.1.11.2.4.3.13.1} }  (ttl 106, id
25600, len 265)

08:33:58.481328 12.155.155.17.50343 > my.own.net.254.161:  [udp sum ok]
{ SNMPv1 { GetNextRequest(220) R=0  .1.3.6.1.2.1.1.1 .1.3.6.1.2.1.1.3
.1.3.6.1.2.1.1.5 .1.3.6.1.2.1.2.2.1.6 .1.3.6.1.2.1.4.20.1.1
.1.3.6.1.2.1.25.3.2.1.3 .1.3.6.1.4.1.11.2.3.9.1.1.7
.1.3.6.1.4.1.11.2.3.9.5.1.3 .1.3.6.1.4.1.11.2.4.3.8.3.2
.1.3.6.1.4.1.11.2.4.3.8.3.3 .1.3.6.1.4.1.11.2.4.3.10.7
.1.3.6.1.4.1.11.2.4.3.10.13 .1.3.6.1.4.1.11.2.4.3.13.1} }  (ttl 109, id
35213, len 265)

02:47:09.174876 216.236.156.62.48249 > my.own.net.1.161:  [udp sum ok] {
SNMPv1 { GetNextRequest(220) R=0  .1.3.6.1.2.1.1.1 .1.3.6.1.2.1.1.3
.1.3.6.1.2.1.1.5 .1.3.6.1.2.1.2.2.1.6 .1.3.6.1.2.1.4.20.1.1
.1.3.6.1.2.1.25.3.2.1.3 .1.3.6.1.4.1.11.2.3.9.1.1.7
.1.3.6.1.4.1.11.2.3.9.5.1.3 .1.3.6.1.4.1.11.2.4.3.8.3.2
.1.3.6.1.4.1.11.2.4.3.8.3.3 .1.3.6.1.4.1.11.2.4.3.10.7
.1.3.6.1.4.1.11.2.4.3.10.13 .1.3.6.1.4.1.11.2.4.3.13.1} }  (ttl 106, id
63089, len 265)
```

Trace 3 - Firewall logging to syslog

--<packet is not allowed to go through and redirected>--

```
Aug 12 02:21:35 myhost kernel: redirecting UDP port 161 to port xx
(216.236.156.62:29158 -> my.own.net.0:161)
```

--<The packetfilter has blocked and silent dropped the packet>--

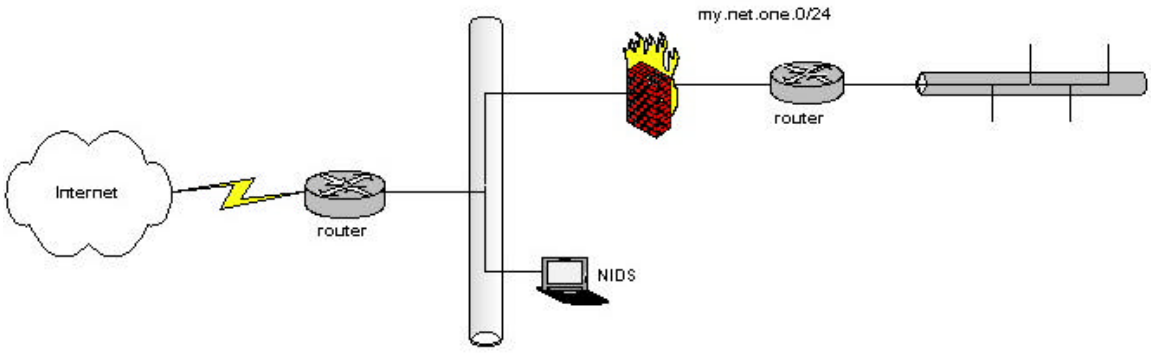
```
Aug 15 02:46:46 myhost /usr/local/etc/local/ipf/ipmon[1595]:
02:46:46.245824 ef0 @38 b 216.236.156.62,48249 -> my.own.net.1,161 PR
udp len 20 265
```

There is no entry for the alert from 12.155.155.17.50343 to my.own.net.254:161 since this packet was allowed to pass. It got however blocked at the next hop.

1. Source of Trace

The source of this trace is made on a C-class address space network which is monitored by my company. The traces are made by a NIDS on the outside of the network and a Tunix firewall logging to syslog. The next diagram shows a simplified representation to illustrate the basic setup and does not necessarily represent the real setup which is far more complex.

Simplified network diagram:



2. Detect was generated by

Snort NIDS version 1.8.6. and a Tunix firewall logging to syslog.
The trace shows a reconnaissance attempt, using the "*public*" community string in a SNMP getNextRequest packet.

3. Probability the source address was spoofed

Source addresses in UDP-packets are likely to be spoofed, since it is usually one-way traffic and often used to launch a DoS attack. In this case, it is not likely that the source address was spoofed because the attacker launches the attack to gather information from the return-packets. It could however be possible that the attacker is somewhere along the return-path where he can sniff and gather the information that is returned along the return-path. In that case, he could have used an unused address belonging to a network up the return-path chain for which he knew that these return packets would be silently dropped, or he could use an existing address for which he knows that the return packets are silently dropped.

The tcpdump log shown below, is a TCP-RST packet which was sent as a response to a stimulus originated on *my.own.net.0* to check if the TTL value could give more insights.

```
16:06:33.371077 216.236.156.62.25 > my.own.net.128.2946: R [tcp sum ok]
0:0(0) ack 4249990451 win 8192 <mss 1460,nop,wscale 0,nop,nop,timestamp
37021175 0> (DF) (ttl 25, id 19352, len 60)
```

The packet shows a TTL value of 25 while the SNMP-packets initiated from this IP, show a TTL value of 106. A TTL value of 106 could indicate a host which is 128-106=22 hops away. The TTL of 25 could indicate a host which is 60-25=35 hops away. It is possible that the TTL value in the SNMP-packets is forged, but if not, the different TTL's confirm the theory that the attacker is somewhere residing along the upstream return-path and that the source address is spoofed.

The initial TTL's used are based on the fact that common initial TTL's are 30, 60, 64(Linux, BSDI, FreeBSD), 128(Windows), and 255 (OpenVMS).

CAN-2002-0515 discusses a design failure of IPFilter in which a port that is filtered by IPFilter will return a RST with a TTL field set to 60, whereas the operating system will return it's default TTL value for a RST.

A window size of 8192 is known to be used by a Solaris 2.2, BSD 4.4 or AIX 3.2, these systems are known platforms for IPFilter.

4. Description of attack

This is a reconnaissance attack by using the *Simple Network Management Protocol* SNMP protocol and is related to CVE CAN-2002-0013. The goal of this attack is to pull information out of the *Management Information Database (MIB)* which is a hierarchical flat file, containing useful information about the device it resides on.

5. Attack mechanism

The trace shows three packets originated from 2 different IP-addresses. The trace with source address 12.155.155.17 has the same odd payload as the packets from source address 216.236.156.62 and is included to support the following: Both sources request the following MIB identifiers which could indicate that it is the same person, operating from 2 different sources, or that this is a known exploit and that they use the same tool:

ObjectID	***Textual name***	***Description***
1.3.6.1.2.1.1.1	sysDescr	Name and version of the device (hardware type and OS)
1.3.6.1.2.1.1.3	sysUpTime	Uptime since the last re-initialization
1.3.6.1.2.1.1.5	sysName	Administratively assigned name for the node (Often the FQDN)
1.3.6.1.2.1.2.2.1.6	ifPhysAddress	The interface's physical address
1.3.6.1.2.1.4.20.1.1	ipAdEntAddr	The IP-address
1.3.6.1.2.1.25.3.2.1.3	hrDeviceDescr	A textual description of the device (manufacturer, revision and/or serial number)
1.3.6.1.4.1.11.2.3.9.1.1.7	Undocumented (HP)	These could belong to HP Jetdirect or to the HP-UX MIB's. Both agents happen to have vulnerabilities through which it is possible to gain administrator privileges
1.3.6.1.4.1.11.2.3.9.5.1.3	Undocumented (HP)	
1.3.6.1.4.1.11.2.4.3.8.3.2	npNpiPaeClass	Returns printer, plottor or xStation for HP devices
1.3.6.1.4.1.11.2.4.3.8.3.3	NpNpiPaeIdentification	Returns the device class(laserjet-IIISI or laserjet-4SI)
1.3.6.1.4.1.11.2.4.3.10.7	NpIpxGetUnitCfgResp2	The Get Unit Config Response in the IPX code
1.3.6.1.4.1.11.2.4.3.10.13	NpIpxRcfgAddress	The 12 octet IPX address
1.3.6.1.4.1.11.2.4.3.13.1	npPortNumPorts	The number of interfaces supported by the device

http://www.mibcentral.com/index.shtml offers a search engine through about 4700 MIB's with over 650.000 OID's. The information above was extracted using this search engine.

The SNMP protocol can be used to maintain and query the MIB of a lot of devices like hubs, switches, routers and printers. Even X-terminals have SNMP agents. The information is protected by passwords, one for querying and one for maintaining the MIB. These passwords are called the *Community String* and have often defaults like "*public"* for read and "*private"* for write access. These passwords are transmitted in plain text (SNMPv1). The traces show usage of the default community string *"public"* and SNMPv1 to query the MIB. An attacker can use this vulnerability in SNMP to gather information and in a second stage, the attacker could reconfigure or shut down devices remotely.

The requested information is focused on specific OID's:

- General system information
  - Hardware type and OS
  - Uptime
  - FQDN
- Interface information
  - Physical address
  - IP address
- Device information like manufacturer, revision and/or serialnumber
- Specific information about HP devices:
  - Two undocumented OID's which could belong to agents which happen to have known vulnerabilities to gain administrator privileges.
  - Class-type of the device
  - IPX information
  - Number of supported interfaces

The attacker is using the SNMP query "*GetNextRequest"*. In general, the GetNextRequest command is used to perform MIB table traversal, and in other cases where the querier does not know the exact MIB name of the object it desires. In that case, the SNMP agent returns the next lexicographically ordered object in the MIB (Stevens[16], Chapter 25).

The first alert is using the IP-address **my.own.net.0** which happens to be the network address. Under normal condition, this UDP packet would be sent to all the hosts residing on this network. All listening SNMP agents on the network could answer with return packets and give information to the attacker about themselves which could be used by the attacker to run known exploits against the device or to learn more about the internals of the network.

The second alert is only used for correlation purposes. It is suspicious that this attacker uses the same odd payload. This could indicate a toolbox or that the attack is performed by the same person/group.

The third alert is targeted at **my.own.net.1**, about three days after the first alert which is most likely an indication for a slow reconnaissance scan. It is most probably one packet in a large scan.

6. Correlations

The SNMP default community string exploit is one of the most critical Internet Security threads which is confirmed by several resources like for example http://www.sans.org/topten.htm.

CVE's regarding this vulnerability:

- CAN-1999-0516
  An SNMP community name is guessable.
- CAN-1999-0517
  An SNMP community name is the default (e.g. public), null, or missing
- CAN-2002-0013
  Vulnerabilities in the SNMPv1 request handling of a large number of SNMP implementations allow remote attackers to cause a denial of service or gain privileges via (1) GetRequest, (2) GetNextRequest, and (3) SetRequest messages, as demonstrated by the PROTOS c06-SNMPv1 test suite.
  NOTE: It is highly likely that this candidate will be SPLIT into multiple candidates, one or more for each vendor. This and other SNMP-related candidates will be updated when more accurate information is available.

http://www.sans.org/newlook/resources/IDFAQ/SNMP.htm describes the SNMP vulnerability to gather information in detail.

This detect with this specific payload has not been reported yet, but this SNMP vulnerability used to gather information using the default community strings is well known.

According to http://www.dshield.org/ipinfo.php?ip=216.236.156.62 An E-mail message was sent to rsehorbach@newcom.com on 2002-08-14 08:57:40 but the message was bounced. This particular IP was accountable for 811 recorded attacks, all against port 161.

7. Evidence of active targeting

This attack is for reconnaissance purposes and probably part of a very large scan so there is no evidence of active targeting.

8. Severity

severity = (criticality + lethality) - (system countermeasures + network countermeasures)
Each element is ranked in a scale from 1-5, with 1 being low and 5 being high.

Criticality: 5

The targeted systems are SNMP agents using default community strings. The traffic which caused the alerts was mainly used for reconnaissance but when

successful, the next step would be setting MIB identifiers with the default community string, which could cause a DoS to the entire network behind a router.

Lethality: 2

This particular traffic is for reconnaissance only and does not cause any disturbance.

System countermeasures: 4

All devices with SNMP agents have this function disabled.

Network countermeasures: 4

SNMP traffic is blocked by firewalling devices at all network-levels.

Severity = (5 + 2) - (4 + 4) = -1

9. Defensive recommendation

Defenses are fine since the attack was blocked and logged by the firewall. In general, systems should not run SNMP agents if not necessary. Running SNMP agents on devices should not be equipped with the default community strings and should only allow SNMP traffic from authorized internal hosts. SNMP traffic coming from the Internet should be blocked by default. If SNMP is necessary, than it is recommended to use SNMPv3 which has better security regarding authentication, privacy, authorization and access control.

10. Multiple choice test question

The log below shows detailed information regarding a *Simple Network Management Protocol* (SNMP) packet. This packet is the only SNMP packet among other various packets that is captured in a tcpdump binary formatted file named **'dump.log'** in it's native binary state.

```
02:47:09.174876 216.236.156.62.48249 > my.own.net.1.161:  [udp sum ok] {
SNMPv1 { GetNextRequest(220) R=0  .1.3.6.1.2.1.1.1 .1.3.6.1.2.1.1.3
.1.3.6.1.2.1.1.5 .1.3.6.1.2.1.2.2.1.6 .1.3.6.1.2.1.4.20.1.1
.1.3.6.1.2.1.25.3.2.1.3 .1.3.6.1.4.1.11.2.3.9.1.1.7
.1.3.6.1.4.1.11.2.3.9.5.1.3 .1.3.6.1.4.1.11.2.4.3.8.3.2
.1.3.6.1.4.1.11.2.4.3.8.3.3 .1.3.6.1.4.1.11.2.4.3.10.7
.1.3.6.1.4.1.11.2.4.3.10.13 .1.3.6.1.4.1.11.2.4.3.13.1} }  (ttl 106, id
63089, len 265)
```

Which command has been used to display this detailed information regarding this packet?

a) snort -dvr dump.log 'udp and port 161'
b) snort -devr dump.log 'udp and port 161'
c) tcpdump -nvr dump.log 'udp and port 161'
d) tcpdump -nvvr dump.log 'udp and port 161'

Answer: c
The newer versions of tcpdump have knowledge about some application data and SNMP is one of them. Answer '*d*' would give even more information. (tcpdump version used is 3.6.3)

Assignment 3 – “Analyze This”-Scenario -

Security Audit for “University”

Executive Summary

This audit is in response to the request from “University” and is the result of analyzing 6 consecutive days of data that was generated by a *Snort Network Intrusion Detection System*, configured with a fairly standard ruleset. The ruleset is extended with some customized additions. The exact content of the ruleset used is unknown. The internal *Security Policy* and network topology are also not provided by “University” in the interest of expediency.

The data that was analyzed, proofs the existence of Trojan activity on some internal servers which is further explained in this audit. There is also evidence of bandwidth consuming applications like IRC-channels, Chat-sessions and file sharing applications. Internal servers are also used for distributed file-access without satisfying security measurements.
Restricting these services will not only enhance security but will also improve network performance.

A lot of network devices are administered by using SNMPv1 and the default community string for reading the configuration of these devices which is not recommended since potential attackers can gain a lot of information regarding the internal organization

Recommendations.
The Snort NIDS produces a lot of noise which could lead to information overload and real important events could easily be overlooked because of this noise. It is recommended that the ruleset will be tuned. If however the Snort NIDS will also be used as a reporting tool concerning traffic flow, then it is recommended to run another Snort NIDS, which performs this task and thus remaining another NIDS to monitor on vulnerabilities and unwanted traffic.

In general, it is recommended to patch all systems, especially the servers, with the latest security patches available on a regular basis. Anti-virus software with the latest signature database should run on all hosts, especially the hosts involved with IRC channels and MSN Messenger. Updates of the Anti-virus signatures should be scheduled with short intervals.

If not already available, procedures and policies should be implemented addressing the use of the high risks and bandwidth consuming application. This policy should also cover potential leakage of certain information which should not be distributed along he Internet.

The use of SNMPv1 to manage network devices should be very restrictive and avoided whenever possible. The usage of the default community strings should not occur.

List of Analyzed Files

The data analyzed consists of the 6-day period between June 11, 2002 through June 16, 2002. The *Out Of Specs* (OOS) data for June 16 was not available, therefore, the data for June 17 was supplied. The data included Snort Alert reports, Snort Scan reports and Snort OOS reports.

Table A - List of analyzed files

Alert	***Scan***	***OOS***
alert.020611.gz	scans.020611.gz	oos_Jun.11.2002.gz
alert.020612.gz	scans.020612.gz	oos_Jun.12.2002.gz
alert.020613.gz	scans.020613.gz	oos_Jun.13.2002.gz
alert.020614.gz	scans.020614.gz	oos_Jun.14.2002.gz
alert.020615.gz	scans.020615.gz	oos_Jun.15.2002.gz
alert.020616.gz	scans.020616.gz	oos_Jun.17.2002.gz
Total: 780,438 entries	Total: 2,129,956 entries	Total: 35 entries

The number of entries as mentioned in *Table A*, are the number of raw entries counted after decompressing the files and checking the files for anomalies like corrupted lines. A checksum in regard to the files is not available. For future use, it is recommended that checksums are provided, since these files are distributed through a public server over the Internet. The version of the Snort NIDS which generated the logs, and the contents of the used ruleset are unknown. The generated alerts are compared against the ruleset of Snort version 1.8.6, to determine whether a default rule generated the alert, or a custom made rule.

List of Detects

Analyses of the Alerts log files

The set of 6 alert-files was concatenated into one large file. After that, all the portscan-alerts are subtracted because these are available in a set of separate scan-files.
The result of the subtraction was, that the totaled alerts-file was reduced with all the spp_portscan noise which was 65.15 percent of all alerts.

Table B - Percentage of portscans in Alerts-file

Entries	***Number of entries***	***Percentage of total***
Alerts without spp_portscan entries	272045	35.85%
Alerts only spp_portscan entries	508393	65.15%
All alerts	780438	100.00%

The *alerts without spp_portscan* entries are used for the alerts-analyze process and is further referred to as *alerts.* These alerts are split into three area's: ***Local (L), External-outbound (O)*** and ***External-inbound (I)*** traffic.

- Local: Source-IP and destination-IP are both local.
- External-outbound: Source-IP is local and destination-IP is not local.
- External-inbound: Source-IP is not local and destination-IP is local.

Dividing the alerts into these three areas is necessary to obtain a good impression about what is happening at "University" network. *Table C - Alerts Summary* is sorted by *"Alerts total"*.

The *"Alerts total"* column is divided into the three area's as mentioned above. The rank number preceding the *area-sub-total* is the rank number the entry has in it's specific area (**L**, **O** or **I**). This table includes only 8 entries, being all entries until all top 3 alerts of every area is included. The complete table with all reported alerts can be found in *appendix A*. The alerts of *Table C* are all described and analyzed.

Table C - Alerts Summary (no scans - excerpt until top 3 of all three areas included)

Alerts Total	***Rank***	***Alerts Local (L)***	***Rank***	***Alerts Out-bound (O)***	***Rank***	***Alerts In-bound (I)***	***Description of the Alert***	***Severity***
59224	1	59224					SMB Name Wildcard	noise
52002	2	52002					SNMP public access	medium
46878	7	383	1	43746	4	2749	spp_http_decode: IIS Unicode attack detected	high
28111					1	28111	MISC Large UDP Packet	medium/high
27142	3	27141	20	1			ICMP Echo Request L3retriever Ping	noise
21959			2	21959			INFO Possible IRC Access	medium
8471			3	4154	3	4317	INFO MSN IM Chat data	medium
4906					2	4906	AFS - Off-campus activity	low

SMB Name Wildcard

Snort rule and description
The Snort rule generating these alerts is a custom made rule. In terms of Intrusion Detection, this rule generates a lot of noise on a network which works with NetBIOS name resolution since this is normal NetBIOS name resolution behavior.

Alert stimulus and traffic analyses
All alerts reported concern only traffic between local machines. This indicates that either the rule which generates the alerts is defined to monitor only local addresses with the SMB Name Wildcard signature, or that all traffic with this signature is being monitored but that the perimeter devices are blocking incoming traffic and that there is no outgoing

traffic. The amount of these alerts indicate the usage of file sharing between Windows hosts or the use of Samba.

The largest part of these alerts (53777 alerts = 90%) is between hosts on the subnet 130.85.152.0/24 and 3 specific hosts, which could indicate normal traffic on this network. The next largest part is originating from 1 host (4147 alerts = 7%).

Hosts on Network (A)	***Specific host (B)***	***# Alerts from (A) → (B)***	***# Alerts from (B) → (A)***	***Total # Alerts***
130.85.152.0/24	130.85.11.5	1416	1445	2861
130.85.152.0/24	130.85.11.6	11503	11658	23161
130.85.152.0/24	130.85.11.7	13776	13979	27755
130.85.0.0/16	130.85.5.89	308	3839	4147

More in-depth details regarding SMB Name Wildcard is described by Alexander Bryce at http://www.sans.org/newlook/resources/IDFAQ/port_137.htm

Recommendation
It is recommended that the Snort ruleset is reconfigured to alert only SMB Name Wildcard traffic when a non local address is involved. As for now, the rule being used is only generating noise from an Intrusion Detection perspective if this traffic is considered normal activity.

SNMP public access

Snort rule and description
The Snort rule generating these alerts is probably coming from the standard rulesets and known as SID 1411 and SID 1412. The severity is categorized as *"attempted-recon"*. All alerts reported have their origin and destination on the local network, but are all triggered due the usage of the well known default community-name *"public"*. The Simple Network Management Protocol (SNMP) is used to manage network-devices from specific management-hosts.

Alert stimulus and traffic analyses
There are 20 different originating hosts (al local) as listed in *table D* (to 150 different destinations). Since the default community-name to read the MIB of devices through SNMP is used (*"public"*), it is not unlikely that there has been some kind of abusive usage.

Table D - Alerts generated from source-IP

# Alerts	***Src-IP***	***# Alerts***	***Src-IP***	***# Alerts***	***Src-IP***	***# Alerts***	***Src-IP***
12105	130.85.70.177	4362	130.85.88.145	444	130.85.88.212	80	130.85.186.10
8773	130.85.88.181	4354	130.85.88.207	394	130.85.88.225	51	130.85.88.176
4766	130.85.150.198	4335	130.85.88.136	358	130.85.153.178	28	130.85.183.11
4414	130.85.88.203	2155	130.85.150.245	152	130.85.91.74	2	130.85.150.114
4372	130.85.88.159	729	130.85.88.251	127	130.85.153.191	1	130.85.6.45

The top 10 critical Internet security threats at http://www.sans.org/topten.htm includes this vulnerability and is known as CVE entries:

- CAN-1999-0517 - Default or blank SNMP community name (public)
- CAN-1999-0516 - Guessable SNMP community name
- CAN-1999-0524, CAN-1999-0186 - Hidden SNMP community strings

Recommendation
It is recommended that the use of SNMP is reduced whenever possible, or otherwise use non-default community-strings and SNMPv3. There are no alerts from or to external addresses which could indicate than only local traffic is being monitored or that perimeter devices block incoming SNMP-traffic. This does not exclude that there are no internal compromised hosts used to perform further reconnaissance by an attacker.
The hosts in *Table D* should be checked against the list of known management-hosts and the 150 destinations should be patched and validated against the list of manageable devices through SNMP. If one of these management hosts is compromised then there may be unauthorized control of manageable clients.

spp_http_decode: IIS Unicode attack detected

Snort rule and description
This alert is generated by the Snort http_decode preprocessor which normalizes HTTP requests by converting any %XX character substitutions to their ASCII equivalent. These Unicode attacks are used by Code Red, Nimda and Sadmind to traverse and escape from the directory used by IIS web-servers.

Alert stimulus and traffic analyses
The alert is generated because of the occurrence of unicode characters in HTTP-requests. This vulnerability is described at http://www.kb.cert.org/vuls/id/111677 and known under CVE-2000-0884. The targeted ports reported are *45,149* alerts to port 80 and *1,729* alerts to port 8080. This could indicate that the http_decode preprocessor is configured to alert on these 2 ports only.

Inbound traffic.

Inbound traffic is originated from the following external source-IPs:

# Alerts	*Source - IP*
10	130.13.107.28
7	130.13.109.53
14	130.13.140.18
9	130.13.166.73
20	130.13.77.117

# Alerts	*Source - IP*
9	202.28.38.209
5	207.230.107.168
7	211.90.223.111
26	213.93.221.118
25	217.167.171.49

# Alerts	*Source - IP*
453	217.225.209.131
12	217.80.93.79
490	217.82.225.167
315	24.101.140.253
12	61.243.8.61

# Alerts	*Source - IP*
5	64.230.87.107
614	65.92.145.85
440	67.81.229.120
276	80.11.161.54

The *targeted internal hosts* are on the following subnets:

# Alerts	*Subnet/Host*	*# Different hosts*
1413	130.85.150.0/24	17 hosts
24	130.85.151.114	1 host
133	130.85.153.0/24	2 hosts

# Alerts	*Subnet/Host*	*# Different hosts*
1111	130.85.5.0/24	5 hosts
68	130.85.88.0/24	3 hosts

Outbound traffic.

The *outbound* connections are originated from the subnets:

# Alerts	*Subnet/Host*	*# Different hosts*
1431	130.85.150.0/24	4 hosts
611	130.85.151.0/24	3 hosts
1033	130.85.152.0/24	20 hosts

# Alerts	*Subnet/Host*	*# Different hosts*
32701	130.85.153.0/24	59 hosts
7970	130.85.88.0/24	12 hosts

Information regarding the destinations of these *outbound* connections:

- Unique hosts targeted: 496 targets.
- When dropping the 4th octet (C-class): 193 targets.
- When dropping the 3rd and 4th octet (B-class): 109 targets.
- When dropping the last 3 octets (A-Class): 27 targets.
- The most interesting range is *211.aa.bb.cc*. There are 311 different hosts in this range which are being targeted by hosts from within the "University"-network.

Local traffic.

The local alerts originate from 3 particular hosts:

# Alerts	*Host*
379	130.85.10.89

# Alerts	*Host*
1	130.85.152.180

# Alerts	*Host*
3	130.85.83.90

The destinations of these local alerts are:

Subnet/Host	*# Different hosts*
130.85.88.0/24	3 hosts
130.85.11.4	1 hosts
130.85.152.0/24	20 hosts

Subnet/Host	*# Different hosts*
130.85.150.0/24	18 hosts
130.85.153.0/24	2 hosts
130.85.5.0/24	5 hosts

The tables above indicate that the “University” network is attacked by using the unicode vulnerability from the outside. It is hard to say if the targeted hosts are patched, but there are however strong indications that a lot of internal hosts, especially on the **130.85.153.0/24** subnet are already infected by one of the worms as mentioned before and are trying to spread.

Recommendation
It is recommended that *all originating* hosts on the local network are checked whether they are compromised (they surely are) and than these compromised hosts should be cleaned and patched. The perimeter devices should be setup in a way that they block this kind of hostile traffic. The URL http://www.incidents.org/react/ has several links with useful information regarding this vulnerability.

MISC Large UDP Packet

Snort rule and description
This alert has the highest number of entries (rank 1) regarding inbound traffic. The rule which is generating this alert, is probably from the basic ruleset and known as SID 521. The rule triggers on incoming UDP packets with a payload of more than 4000 bytes. The alert is classified as "*bad-unknown"*. In general, since this alert is caused by UDP packets, the source addresses can easily be forged.

The underground programmer known as *"Mixter"* describes these large packages as a possible DDoS attack since Stateful UDP sessions are normally using small UDP packets. (http://www.digitaltrust.it/arachnids/IDS247/research.html)

Alert stimulus and traffic analyses
It is necessary to have (some) of the payload for detailed investigation and value these alerts properly. For now, only some conclusions about the origins and destinations can be made. To determine if this was an attack or benign traffic, it is also important to know what was the stimulus. Since not all packets are being captured, this cannot be determined accurately from the given log-files. However, it is possible to gather all traffic between the hosts and ports involved. There are only 16 connections which generate this alert.

# Alerts	*Source IP*	*Destination IP*	*# Ports*
12708	140.142.8.72	130.85.153.157	11
5810	202.102.249.118	130.85.88.140	8
3283	140.142.8.71	130.85.153.159	6
1642	211.63.185.15	130.85.153.179	7
1107	211.63.185.21	130.85.153.179	8
1023	**10.16.2.4**	130.85.150.209	8
716	202.210.163.74	130.85.152.21	29
633	208.252.239.125	130.85.150.209	2

# Alerts	*Source IP*	*Destination IP*	*# Ports*
552	211.233.77.23	130.85.153.115	52
372	205.188.244.227	130.85.151.108	5
125	**10.16.2.1**	130.85.150.209	2
49	**207.189.78.251**	130.85.152.186	2
47	202.123.219.153	130.85.153.187	12
27	202.103.102.114	130.85.153.45	3
16	211.233.45.59	130.85.88.222	6
1	210.220.161.102	30.85.153.203	1

The column *"# Ports"* summarizes the number of destination ports that were targeted. The Source-IP's where checked against the alert file and all source-IP's but only one was reported in this alert. The exception was source-IP 207.189.78.251 which was also alerted on inbound alert rank 2 (*AFS - Off-campus activity*), connecting to 130.85.153.210 on port 7001 (once).

At least two IP numbers indicate spoofing or misconfigured hosts on the local network since these rfc1218 (http://www.rfc-editor.org/rfc/rfc1918.txt - private range address-space) should not pass the perimeter devices:

- IP 10.16.2.1 which targeted port 0 (3 times) and port 4145 (122 times)
- IP 10.16.2.4 which targeted port 0 (6 times), port 20583 (once), port 3238 (613 times), port 3257 (147 times), port 3630 (90 times), port 40716 (once), port 4133 (164 times) and port 7000 (once).

The targeted host (130.85.150.209) was also targeted by 208.252.239.125 on ports 2259 (294 times) and 2283 (339 times) This host 208.252.239.125 was the only source-address which did not try to connect to port 0.

There where 141 distinct destination ports out of 162 reported ports used of which 136 ports were only addressed once. The ports which are targeted by more than 1 host are port 0 (15 hosts), port 1588 (4 hosts), port 7000 (3 hosts) and port 25970 (2 hosts).

The following responses were found because of the Snort rule SID 410 which triggers on "ICMP Fragment Reassembly Time Exceeded":

# Alerts	*From*	*To*
4	130.85.150.209	208.252.239.125
11	130.85.152.21	202.210.163.74
5	130.85.153.115	211.233.77.23

# Alerts	*From*	*To*
3	130.85.153.179	211.63.185.21
3	130.85.88.140	202.102.249.118
13	130.85.88.222	211.233.45.59

This indicates that the local hosts did process the UDP packets but that reassembly did not succeed. This could be caused by forged packets.

Practical #525 at http://www.giac.org/practical/Tod_Beardsley_GCIA.doc sent in by Tod A. Beardsley, discovered that Windows Media Services sometimes relies on large UDP packets to deliver streaming content.

Recommendation
It is recommended to investigate and tune these alerts. The use of destination (and source) port 0 indicates some kind of reconnaissance and could be blocked by the perimeter devices. There are also indications that the local hosts identified are using applications which depend on these large UDP packages. Misuse of these hosts should be investigated.

ICMP Echo Request L3retriever Ping

Snort rule and description
The Snort rule generating these alerts is probably coming from the standard ruleset and known as SID 466. The severity is categorized as "*attempted-recon"* and it alerts on ICMP echo request (type 8, code 0) packets with a content of "ABCDEFGHIJKLMNOPQRSTUVWABCDEFGHI" at the start of the payload.

The *"L3 Retriever 1.5"* security scanner is a legitimate tool for security assessments, distributed by Symantec to discover network components, which has the same signature. Since this traffic depends on ICMP packets, the source IP address could be easily forged, but for reconnaissance, it is necessary that the return packet arrives or at least passes on its way back to the attacker.

According to ArachNIDS (http://www.digitaltrust.it/arachnids/IDS311/event.html) event IDS311, there are reported incidents where legitimate traffic may cause an intrusion detection system to raise "false positive" alerts for this event.
Begin quote

> *"This type of ICMP ping seems to be also generated by (plain) Win2K host talking to Win2K domain controllers." --nnposter*

End quote.

Alert stimulus and traffic analyses
There was only 1 outbound packet reported and this packet was from local IP 130.85.88.140 to IP 202.119.37.5. There are no other alerts found for IP 202.119.37.5, neither in the Alert, Scans or OOS logs. This has probably been a misconfiguration.

All other alerts are local and can be categorized into 196 connections. These connections are summarized by subnet and destination. There are only 12 different destinations

Information according the 196 different connections:

Subnet	***# hosts on subnet***	***Destinations (#alerts)***			
130.85.152.0/24	55	130.85.11.5 (1433)	130.85.11.6 (11528)	130.85.11.7 (13832)	
130.85.150.0/24	12	130.85.5.4 (181)	130.85.5.35 (12)	130.85.5.3 (2)	
130.85.88.0/24	10	130.85.5.4 (79)	130.85.130.187 (28)	202.119.37.5 (1)	
130.85.151.0/24	4	130.85.5.4 (14)	130.85.5.35 (5)		
130.85.153.0/24	2	130.85.5.4 (12)			
130.85.0.0/16 (remaining)	9	130.85.5.96 (11)	130.85.5.92 (2)	130.85.5.46 (1)	130.85.5.64 (1)

Recommendation
Given the fact that this could be legitimate Windows 2000 traffic, it is recommended to fine-tune this rule in regard to existing WINDOWS 2000 domain controllers, so that there will be less noise generated. This fine-tuning consists of configuring pass-rules for legitimate Windows 2000 domain controllers which are probably identical to the destinations mentioned in the table above.

INFO Possible IRC Access

Snort rule and description
The snort rule generating these alerts is probably a local defined rule which alerts on traffic to a port ranging from 6666 to 7000. The ruleset 1.8.6 is more specific with this type of traffic to these ports and would generate more meaningful alerts by looking into the content of the packets. These 1.8.6. rules have the following characteristics:

Alert message	***Contents in the packet must include the string:***		
CHAT IRC nick change	"NICK "		
CHAT IRC DCC file transfer request	"PRIVMSG "	" \:.DCC SEND"	
CHAT IRC DCC chat request	"PRIVMSG "	" \:.DCC CHAT chat"	
CHAT IRC channel join	"JOIN \: \#		
CHAT IRC message	"PRIVMSG "		
CHAT IRC dns request	"USERHOST"		
CHAT IRC dns response	"\:"	" 302 "	"=+"
CHAT IRC EXPLOIT topic overflow	"|eb 4b 5b 53 32 e4 83 c3 0b 4b 88 23 b8 50 77|"		
CHAT IRC EXPLOIT Ettercap parse overflow attempt	"PRIVMSG nickserv IDENTIFY"		

A good start to find more about IRC could be http://www.irchelp.org. This site also maintains IRC-server lists which could be useful. The index of these lists can be found at http://www.irchelp.org/irchelp/networks/servers/index.html. These lists also contain port information which reveals that monitoring the port range 6666-7000 is definitely not sufficient since there are a lot of other ports that are used like 6660-7000, 5555, 7770-7775, 7777, 8000 and 9000.

IRC servers make a living out of the IRC services and most of them put a lot of effort in protective measures, but still, the IRC channels spread viruses and people are often unaware of specific commands they type which makes them vulnerable to hackers. Social Engineering is another attack which is performed through the IRC channels.

There are 35 CVE entries or candidates that match a search on the keyword "IRC" and a lot of them describe buffer overflows, bypasses on firewalls and gaining control over the client-host.

The CVE entries are:

CVE-1999-0672	CVE-2000-0594	CVE-2002-0060	CAN-2000-0785	CAN-2002-0006
CVE-1999-0679	CVE-2000-0601	CVE-2002-0197	CAN-2000-0857	CAN-2002-0382
CVE-1999-0939	CVE-2000-0661	CAN-1999-0220	CAN-2000-1102	CAN-2002-0425
CVE-1999-0968	CVE-2000-0787	CAN-1999-0255	CAN-2000-1150	CAN-2002-0593
CVE-1999-1351	CVE-2001-0050	CAN-1999-0645	CAN-2000-1151	CAN-2002-0928
CVE-2000-0183	CVE-2001-0274	CAN-1999-0661	CAN-2000-1152	CAN-2002-0983
CVE-2000-0586	CVE-2001-1056	CAN-1999-1228	CAN-2001-0177	CAN-2002-0984

Alert stimulus and traffic analyses
There are only 20 different connections who generates these alerts:

Source-IP	*Destination-IP*	*# Alerts*
130.85.151.90	64.246.34.181	6650
	66.62.70.248	7459
	66.28.132.168	7814
130.85.150.165	*207.68.167.253*	14
130.85.88.151		4
130.85.153.202		1
130.85.153.150		1
130.85.153.71		1
130.85.153.197	*209.151.249.50*	1
130.85.153.162	216.152.64.62	1

Source-IP	*Destination-IP*	*# Alerts*
130.85.88.165	*151.189.12.24*	2
	195.159.0.90	2
	203.121.68.222	2
	151.189.12.20	1
	212.110.161.45	1
	64.212.171.241	1
130.85.152.244	*64.212.171.241*	1
130.85.88.143	*211.192.139.41*	1
130.85.153.203	*211.192.139.41*	1
	211.63.185.161	1

After digging deeper into the alerts for correlated connections of these possible IRC access alerts, following related alerts showed up which could all confirm the use of IRC. The IP-numbers in *italic* have no correlated data.

Alert	*Source-IP*	*Destination-IP*	*port*	*# Alerts*
IRC evil - running XDCC This could indicate a chat or file transfer request which is not advisable.	130.85.151.90	64.246.34.181	6667	7
		66.28.132.168	6667	14
		66.62.70.248	6667	58
INFO - Possible Squid Scan This is 'normal' behavior for IRC-servers to test if a IRC client is not behind a squid-server.	195.159.0.90	130.85.88.165	3128	3
	216.152.64.62	130.85.153.162	3128	1
	66.28.132.168	130.85.151.90	3128	2
	66.62.70.248	130.85.151.90	3128	5
SCAN Proxy attempt This is 'normal' behavior for IRC-servers to test if a IRC client is not behind a proxy-server.	195.159.0.90	130.85.88.165	1080	4
			8080	3
	216.152.64.62	130.85.153.162	1080	1
			8080	1
	66.28.132.168	130.85.151.90	1080	4
			8080	2
	66.62.70.248	130.85.151.90	1080	10
			8080	5

Recommendation
It is recommended to apply the most recent ruleset whenever the IRC services are needed and to patch the IRC-clients very well. Also training of the users is very important. The rule which generates the current alerts, is to noisy. The top 3 of destination-IP's for this alert is also the top 3 of all outbound destinations.

INFO MSN IM Chat data

Snort rule and description
This is probably a custom made Snort rule, since it is either activated by outbound traffic on destination port 1863 and inbound traffic from port 1863 or, more likely, it triggers on hosts in the network range 64.4.12.0/23 which is registered by MS Hotmail (NETBLK-HOTMAIL). It reports only on connections and from an intrusion analyses perspective, these alerts can be classified as noise.

Alert stimulus and traffic analyses
There are 63 hosts on net 64.4.12.0/24 and 4 hosts on net 64.4.13.0/24 targeted by 38 different internal hosts. These 38 internal hosts are:

# Alerts	*Local host*
1334	130.85.150.242
855	130.85.150.232
652	130.85.153.142
630	130.85.153.157
589	130.85.153.107
587	130.85.153.46
491	130.85.153.45
448	130.85.150.241
345	130.85.88.201
311	130.85.88.154

# Alerts	*Local host*
217	130.85.88.215
201	130.85.151.55
196	130.85.150.165
184	130.85.153.110
183	130.85.88.146
139	130.85.153.190
137	130.85.153.159
134	130.85.88.229
120	130.85.88.151
107	130.85.88.140

# Alerts	*Local host*
101	130.85.88.236
101	130.85.88.165
91	130.85.153.167
55	130.85.153.109
53	130.85.153.111
46	130.85.150.113
42	130.85.151.64
35	130.85.150.59
24	130.85.153.125

# Alerts	*Local host*
14	130.85.153.168
14	130.85.150.133
12	130.85.88.220
7	130.85.150.206
5	130.85.88.143
5	130.85.153.180
2	130.85.153.120
2	130.85.150.46
2	130.85.150.143

The traffic itself is benign in nature, but is susceptible for virus and Trojans issues, especially new ones. The targeted MSN hosts triggered no other alerts.

Recommendation
It is recommended that this traffic is blocked by the perimeter devices, however when these MSN Messenger Services are necessary, all hosts involved should be patched and run up to date anti-virus software with the most recent virus signatures. Training of the users and security awareness is also preferable. More info about MSN Instant Messenger (IM) can be found at http://messenger.msn.com.

AFS - Off-campus activity

Snort rule and description
These alerts are generated by a custom made Snort rule which most likely is defined to trigger on inbound traffic destined for port 7001, the default port for AFS-server. AFS (Andrew File System) enables distributed file sharing and has several levels of security. The most secure configuration is when AFS is configured to use Kerberos authentication and *mutual authentication*. Mutual authentication is the means through which parties prove their genuineness and works with generated session keys. It is also possible to generate IP based Access Control Lists. More info about AFS can be found at http://www.transarc.com and http://www.openafs.org

There is an integer overflow bug in the SUNRPC-derived RPC library used by OpenAFS that could be exploited to crash certain OpenAFS servers (volserver, vlserver, ptserver, buserver) or to obtain unauthorized root access to a host running one of these processes. OpenAFS Security Advisory records this overflow bug as "ID 2002-001- 1.0-1.2.5, 1.3.0-1.3.2 xdr_array integer overflow".

Alert stimulus and traffic analyses
There were 91 different internet hosts which connected to 36 different internal hosts, causing 147 different connections. In most cases, the source port was 7000 and the destination port was 7001.

Exceptions to this destination port 7001 are the following connections which generated also traffic from another source port:

Source - IP	***Source - port (A)***	***# alerts on A***	***# alerts on source port 7000***	***Destination - IP***	***Dest.-port***
12.151.57.37	7003	4	2125	130.85.88.245	7001
128.8.70.9	7003	2	6	130.85.153.168	7001
18.181.0.19	7003	2	71	130.85.153.189	7001
18.181.0.22	7003	1	0	130.85.153.189	7001
18.181.0.23	7003	6	108	130.85.153.189	7001
66.28.225.156	512	1	185	130.85.151.95	7001

The 36 internal hosts were visited by the following number of external hosts:

# Alerts	***# Hosts***	***Destination - IP***
2129	1	130.85.88.245
640	7	130.85.151.95
297	7	130.85.152.19
297	21	130.85.153.168
291	12	130.85.151.66
222	9	130.85.150.232
219	1	130.85.153.46
197	12	130.85.153.165
188	3	130.85.153.189
71	10	130.85.153.142
58	8	130.85.153.145
44	3	130.85.153.169

# Alerts	***# Hosts***	***Destination - IP***
40	4	130.85.152.216
36	3	130.85.153.179
28	6	130.85.150.240
27	4	130.85.153.176
23	3	130.85.153.159
17	1	130.85.153.115
15	4	130.85.88.201
12	1	130.85.152.150
8	1	130.85.88.183
7	3	130.85.150.46
7	2	130.85.88.235
5	3	130.85.152.164

# Alerts	***# Hosts***	***Destination - IP***
5	1	130.85.152.246
4	2	130.85.152.165
4	1	130.85.153.148
3	1	130.85.88.140
3	1	130.85.153.187
2	1	130.85.153.216
2	1	130.85.152.48
1	1	130.85.153.210
1	1	130.85.153.157
1	1	130.85.152.21
1	1	130.85.152.157
1	1	130.85.150.209

There is a lot of traffic to the AFS ports. When condensing the traffic on network boundary’s, it turned out that the connections come from a few network ranges which could indicate that traffic originated at these networks is allowed.

Net	*# hosts*	*Destination*	*# Alerts*
211.239.164.0/24	24	130.85.153.165	197
		130.85.153.168	236
		130.85.153.179	9
		130.85.88.201	15
63.250.205.0./24	14	130.85.150.232	1
		130.85.151.66	82
		130.85.152.19	1
		130.85.153.142	64
		130.85.88.235	7
210.115.150.0/24	2	130.85.153.176	5
211.233.25.0/24	8	130.85.153.169	44
		130.85.153.115	17
		130.85.153.168	1
		130.85.153.148	4
		130.85.153.216	2
		130.85.152.164	1

Net	*# hosts*	*Destination*	*# Alerts*
63.250.219.0/24	9	130.85.150.232	221
		130.85.150.240	28
		130.85.152.164	4
		130.85.152.165	4
		130.85.152.19	296
		130.85.152.216	40
		130.85.153.142	6
		130.85.153.145	48
		130.85.153.159	20
66.28.14.0/24	3	130.85.151.66	75
		130.85.151.95	116
210.115.150.0/24	2	130.85.153.176	22
63.208.104.0/24	3	130.85.150.46	7
66.250.64.10/32	1	130.85.151.66	56
		30.85.151.66	124

The remaining connections are sorted by destination and included to check against the allowed AFS usage list:

# Alerts	*Source-IP*	*Destination-IP*
3	202.102.249.118	130.85.88.140
1	208.252.239.125	130.85.150.209
22	64.215.213.238	130.85.151.95
192	195.92.228.141	130.85.151.95
1	202.210.163.74	130.85.152.21
2	64.15.254.25	130.85.152.48
12	64.15.254.25	130.85.152.150
1	63.250.214.130	130.85.152.157
5	63.210.47.79	130.85.152.246
1	202.58.56.172:	130.85.153.142

# Alerts	*Source-IP*	*Destination-IP*
10	213.239.135.175	130.85.153.145
1	63.211.17.234	130.85.153.157
3	140.142.8.71	130.85.153.159
4	28.8.10.207	130.85.153.168
48	129.2.128.54	130.85.153.168
26	218.50.55.40	130.85.153.179
1	211.63.185.21	130.85.153.179
3	202.123.219.153	130.85.153.187
1	207.189.78.251	130.85.153.210

There were also two suspicious connects:

- 219 alerts from IP 10.16.1.40:7000 to IP 130.85.153.46:7001
- 8 alerts from IP 10.16.3.3:7000 to IP 130.85.88.183:7001

Recommendation

It is recommended that when AFS is needed, to use Kerberos and mutual authentication. The alerts indicate that there are *"known"* remote hosts/networks and random hosts which connect. If both type of hosts are required, than it is recommended to split the backend servers to protect some of them (the ones that are used by known hosts) with ACL's. The

Snort ruleset should be adjusted in a way that the allowed hosts are only registered and other hosts are alerted upon to reduce the noise. It seems that the **AFS login** function has been used which is plain text authentication traveling the Internet which is not advisable.

Top Talkers list - Alerts

The top Alert talkers list is composed using the same area's as the alert analyses and without the scan-alerts. The area's are *Local*, *Inbound* and *outbound* traffic.

Table E - Top 10 Alerts talkers -Local

	# Alerts	***Source – IP***
1	13979	130.85.11.7
2	12129	130.85.70.177
3	11658	130.85.11.6
4	8773	130.85.88.181
5	4766	130.85.150.198
6	4414	130.85.88.203
7	4376	130.85.88.159
8	4362	130.85.88.145
9	4355	130.85.88.207
10	4343	130.85.88.136

	# Alerts	***Destination – IP***
1	29663	130.85.11.7
2	27417	130.85.150.195
3	24588	130.85.11.6
4	4820	130.85.150.84
5	3123	130.85.5.96
6	2849	130.85.11.5
7	2828	130.85.5.97
8	2690	130.85.5.127
9	2175	130.85.152.109
10	1647	130.85.113.202

Table F - Top 10 Alerts talkers - Inbound

	# Alerts	***Source - IP***	***Reverse dns-lookup***
1	12708	140.142.8.72	Media-wm-2.cac.washington.edu
2	5813	202.102.249.118	SOA in domain .zz.ha.cn
3	3286	140.142.8.71	Media-wm-1.cac.washington.edu
4	2452	12.151.57.37	SOA in domain .securitas.bz
5	2038	212.179.40.132	station-131.gadot.org.il
6	1642	211.63.185.15	SOA in domain .kornet.net
7	1165	65.92.145.85	HSE-Montreal-ppp337094.sympatico.ca
8	1108	211.63.185.21	SOA in domain .kornet.net
9	1023	10.16.2.4	Private range address
10	834	217.82.225.167	pD952E1A7.dip.t-dialin.net

	# Alerts	***Destination - IP***
1	13093	130.85.153.157
2	5864	130.85.88.140
3	3498	130.85.153.179
4	3373	130.85.153.159
5	2452	130.85.88.245
6	2273	130.85.150.209
7	2070	130.85.88.162
8	1305	130.85.5.96
9	729	130.85.150.232
10	726	130.85.152.21

Table G - Top 10 Alerts talkers - Outbound

	# Alerts	Source - IP
1	22003	130.85.151.90
2	9893	130.85.153.179
3	4683	130.85.153.136
4	3718	130.85.88.201
5	2456	130.85.88.143
6	2239	130.85.153.165
7	2158	130.85.153.163
8	2129	130.85.153.168
9	2090	130.85.153.157
10	1416	130.85.153.115

	# Alerts	Destination - IP	Reverse dns-lookup
1	7828	66.28.132.168	unf.unf.unf.u.nf
2	7517	66.62.70.248	SOA in domain .in-tch.com
3	6657	64.246.34.181	SOA in domain .ev1.net
4	3300	211.210.13.212	SOA in domain .krnic.net
5	2295	211.239.164.163	SOA in domain .gngidc.net
6	2256	211.239.164.180	SOA in domain .gngidc.net
7	1880	211.63.185.30	SOA in domain .kornet.net
8	1749	209.10.239.135	SOA in domain .globix.net
9	1136	211.63.185.26	SOA in domain .kornet.net
10	1060	211.239.123.75	SOA in domain .gngidc.net

Analyses of the OOS logfiles

Out Of Spec (OOS) log files contain the clear text Snort output of those IP packets which do not comply with the RFC TCP/IP specifications. These packets are often crafted or the result of a malfunctioning network device.

The OOS files were dated one day later than the actual contents. The actual range of OOS was from 06/10 until 06/15. There were two entries in the OOS file that contained data which could be the result of an overloaded logging process or manual editing of the data. These entries are:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
06/10-19:06:09.354134 65.42.230.217:1342 -> MY.NET.88.162:0
TCP TTL:115 TOS:0x0 ID:62883  DF
21**RP** Seq: 0x4BE0040   Ack: 0x84883C72   Win: 0x5010
TCP Options => EOL EOL EOL EOL EOL EOL SackOK Opt 21 Opt 21 Opt 21 Opt 21 Opt 21
Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt
21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21 Opt 21
Opt 21 Opt 21 Opt 21 Opt 21 Opt 21
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
06/13-17:54:54.956901 64.4.124.151:4 -> MY.NET.88.165:3193
TCP TTL:113 TOS:0x0 ID:65215  DF
21**R*** Seq: 0x4F50FC7   Ack: 0x31D87D88   Win: 0x5010
TCP Options => EOL EOL EOL EOL EOL EOL SackOK SackOK SackOK EOL Opt 53 Opt 53
Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt
53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53 Opt 53
Opt 53 Opt 53 Opt 53 Opt 53 Opt 53
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The timestamp of the second entry was within the range of scans and alerts logfiles and this timestamp was checked against both files for the amount of entries during that time. There was no evidence found that the NIDS was overloaded during that time.

There were 5 packets originating from 65.65.224.233 to MY.NET.88.162 which have the "DF" and "MF" flags both set. This is an out of spec because they can not be set both in the same packet. GCIA Practical #0489 by Hee So analyzed packets with exactly the same signature (http://www.giac.org/practical/Hee_So_GCIA.doc).

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
06/10-16:59:22.453703 65.65.224.233 -> MY.NET.88.162
TCP TTL:110 TOS:0x0 ID:1553  DF MF
Frag Offset: 0x0   Frag Size: 0x22
91 87 92 50 1E DA B2 02 72 12 2B 94 03 C3 BE DF  ...P....r.+.....
EB 14 B6 86 C4 D1 D8 AC 65 F0 FA 29 E5 2D E9 2C  ........e..).-.,
EF 0A                                            ..
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The IP identification is incremented for every packet, the Fragment Offset is always 0 and the Fragment size is 34 (0x22) bytes. There are no correlations to the other logs since they contained no entries dated 06/10. The source IP does not further show up in the other logs. This could be an improperly functioning network device as stated by Hee So.

The remaining packets had all weird TCP flags combination and the connections were matched against the alerts file which results in the following overview of suspect behavior.

# Alerts	*Alert Code*	*Connection*	*# OOS*	*OOS-SRC*	*OOS-DST*
		IP 64.4.124.151			
1	A	64.4.124.151:3193 -> 130.85.88.165:1269	1	64.4.124.151:0	MY.NET.88.165:3193
14	B	64.4.124.151:3193 -> 130.85.88.165:1269	5	64.4.124.151:3193	MY.NET.88.165:1269
8	C	64.4.124.151:3193 -> 130.85.88.165:1269	1	64.4.124.151:4	MY.NET.88.165:3193
		IP 24.112.58.210			
4	B	24.112.58.210:2656 -> 130.85.150.209:6346	1	24.112.58.210:166	MY.NET.150.209:2656
1	D	24.112.58.210:2656 -> 130.85.150.209:6346	5	24.112.58.210:2656	MY.NET.150.209:6346
		IP 193.6.40.86			
2	E	193.6.40.86:55089 -> 130.85.150.209:6346	2	193.6.40.86:55089	MY.NET.150.209:634
		IP 195.101.94.208			
1	E	195.101.94.208:1107 -> 130.85.150.83:80	1	195.101.94.208:1107	MY.NET.150.83:80
1	E	195.101.94.208:1385 -> 130.85.5.96:80	1	195.101.94.208:1385	MY.NET.5.96:80
1	E	195.101.94.208:2033 -> 130.85.5.95:80	1	195.101.94.208:2033	MY.NET.5.95:80
1	E	195.101.94.208:2102 -> 130.85.5.95:80	1	195.101.94.208:2102	MY.NET.5.95:80
		IP 62.78.169.87			
1	E	62.78.169.87:38498 -> 130.85.150.209:6346	1	62.78.169.87:38498	MY.NET.150.209:6346
1	D	2.78.169.87:38498 -> 130.85.150.209:6346			
		IP 62.99.143.178			
1	E	62.99.143.178:59781 -> 130.85.150.83:80	1	62.99.143.178:59781	MY.NET.150.83:80
		IP 62.99.143.179			
1	E	62.99.143.179:42643 -> 130.85.150.83:80	1	62.99.143.179:42643	MY.NET.150.83:80
		IP 12.217.65.38			
2	B	12.217.65.38:4106 -> 130.85.88.162:1214	1	12.224.236.145:65329	MY.NET.88.162:1214
		IP 12.217.65.38			
0			1	12.217.65.38:0	MY.NET.88.162:4106

Alert-Code A: EXPLOIT x86 setuid 0
Alert-Code B: Null scan!
Alert-Code C: SCAN FIN
Alert-Code D: INFO Inbound GNUTella Connect request
Alert-Code E: Queso fingerprint

IP 64.4.124.151 is definitely scanning the local host 130.85.88.165 for vulnerabilities. The destination ports are 3193 and 1269. Port 1269 is known to be used by the Trojan called Matrix. The scan logs also contain several entries (41 alerts), all with malformed TCP-flag combinations. This IP is referenced in the *Registration Information* section. It is recommended to notify the owner of this IP.

The *Queso fingerprint* which is the most reported alert, is used to determine the OS of the target host. The TCP-flags are denoted by "21S*****". More information is available at http://www.digitaltrust.it/arachnids/IDS29/research.html.

Top Talkers list - OOS

	# Alerts	*Source - IP*	*Reverse dns-lookup*
1	7	64.4.124.151	64-4-124-151.dmt.ntelos.net
2	6	24.112.58.210	CPE00045a7be40b.cpe.net.cable.rogers.com
3	5	65.65.224.233	adsl-65-65-224-233.dsl.spfdmo.swbell.net
4	4	195.101.94.208	x1crawler1-1-0.x-echo.com
5	2	24.120.177.22	cm022.177.120.24.lvcm.com
6	2	193.6.40.86	bazooka.saturnus.vein.hu
7	1	68.80.114.202	pcp01351643pcs.benslm01.pa.comcast.net
8	1	68.50.107.141	bgp01545206bgs.gambrl01.md.comcast.net
9	1	66.25.185.163	cs6625185-163.austin.rr.com
10	1	65.42.230.217	adsl-65-42-230-217.dsl.ipltin.ameritech.net

	# Alerts	*Destination – IP*
1	11	MY.NET.88.162
2	9	MY.NET.150.209
3	7	MY.NET.88.165
4	3	MY.NET.5.96
5	3	MY.NET.150.83
6	2	MY.NET.5.95
7	0	None
8	0	None
9	0	None
10	0	None

Analyses of the Scan logfiles

The Snort NIDS generated 2,129,956 scan log entries which are divided into:

137,411 entries : Inbound scans
291,995 entries : Outbound scan
1,700,550 entries : Scans on the local network

Scans on the Local Network
The top 10 destination ports within the scans on the local network are listed in the next table. The "*# Alerts*"-column shows the total amount of alerts generated to a specific port for one protocol (UDP/TCP).
The "*# Total-Alerts*"-column shows the total number of alerts for all protocols.

# Alerts	*Destination port*	*Type*	*Description*	*# Total-Alerts*
606475	161	UDP	SNMP	606477
135369	7001	UDP	AFS callback	135371
88003	53	UDP	Dns	88021
83819	7000	UDP	AFS	83819
53426	137	UDP	NETBios name service	53428
24453	0	UDP	Unused	24453
21765	7003	UDP	AFS vlserver	21766
16419	1346	UDP	Alta analystics lic. Man. or Ghost multicast	16421
15413	139	SYN	NETBios session service	15442
12031	514	UDP	Syslog	12034

Most of the traffic listed above is normal. Only destination port '0' is definitely none standard traffic. There are 134 internal hosts which generate this traffic which all (except 3 connections) use source-port 0. These 134 hosts should be investigated.

Another anomaly is that when looking at the difference between "# Alerts" and "*# Total-*Alerts" there are 2 hosts which are mainly responsible for these other connections:

From: 130.85.28.2 port 38976	to 130.85.151.90	port 161	SYN ******S*
		port 7001	SYN ******S*
		port 137	SYN ******S*
		port 1346	SYN ******S*
		port 514	SYN ******S*
From: 130.85.253.10 port 44591	to 130.85.88.245	port 161	SYN ******S*
		port 7001	SYN ******S*
		port 137	SYN ******S*
		port 7003	SYN ******S*
		port 514	SYN ******S*

It is recommended to investigate these hosts thoroughly on being compromised or not.

Inbound scans
The top 2 destination ports within the inbound scans are:

49532	alerts to port:	6970	Trojan GateCrasher
24753	alerts to port:	0	Unused

These scans mainly originate from host 12.151.57.37 and the network 205.188.222.0/24 which are also listed in the "*Top Talkers list –Scans*". See also the *Registration Information* section.

The top 10 of internal hosts that were scanned by one external host, ranked on destination port:

IP 12.151.57.37	probed	130.85.88.245	on	7492	ports	(	45027	alerts)
IP 195.92.228.141	probed	130.85.151.95	on	663	ports	(	3328	alerts)
IP 10.16.1.40	probed	130.85.153.46	on	562	ports	(	2454	alerts)
IP 213.239.135.175	probed	130.85.153.145	on	210	ports	(	351	alerts)
IP 211.239.164.43	probed	130.85.153.168	on	185	ports	(	413	alerts)
IP 63.250.205.12	probed	130.85.153.142	on	171	ports	(	388	alerts)
IP 211.239.164.15	probed	130.85.153.168	on	157	ports	(	427	alerts)
IP 63.119.175.56	probed	130.85.88.226	on	155	ports	(	155	alerts)
IP 211.233.25.31	probed	130.85.153.169	on	153	ports	(	388	alerts)
IP 211.239.164.44	probed	130.85.153.168	on	135	ports	(	431	alerts)

Outbound scans
The top 10 destination ports for generated scans leaving the local network are:

# Alerts	***Port***	***Type***
187074	80	SYN
33582	1214	UDP
14700	6667	SYN
11915	6346	SYN
4342	427	UDP

# Alerts	***Port***	***Type***
2230	1214	SYN
1759	443	SYN
1684	6257	UDP
1081	5190	SYN
799	8080	SYN

The huge amount of entries for port 80 mostly originate from hosts on the subnets 130.85.153.0/24, 130.85.152.0/24, 130.85.151.0/24, 130.85.150.0/24 and 130.85.88.0/24 which correlates with the alert analyses of the "*spp_http_decode: IIS Unicode attack detected*" and confirms that these originating hosts are probably compromised.

The alerts to port 1214 are mostly originated from the hosts 130.85.88.162 (24251 alerts) and 130.85.150.113 (9125 alerts) and there are 11223 different targets. This indicates the usage of KaZaa/Morpheous/Grokster filesharing. It is recommended to check the usage of this file sharing against the policy.

The alerts to port 6667 confirm the heavy use of IRC communications as stated in the alert analyses of "*INFO Possible IRC Access*".

The alerts to port 6346 and 6257 indicate the use of Gnutella, BearShare or WinMX file sharing. The use of Gnutella on outbound connections is also reported with the alerts analyses (ranked 7, outbound connections).

The alert to port 5190 indicates the usage of AOL instant messenger.

Top Talkers list - Scans

Table H Top 10 Scan talkers - External source and Internal source

	# Alerts	External source	Reverse dns-lookup
1	45027	12.151.57.37	SOA in .securitas.bz
2	11336	205.188.228.129	mslb6.streamops.aol.com
3	10721	205.188.228.1	mslb1.spinner.com
4	9802	205.188.228.145	mslb7.streamops.aol.com
5	6778	205.188.228.17	mslb2.spinner.com
6	6673	205.188.228.65	mslb4.spinner.com
7	4389	205.188.228.33	mslb3.spinner.com
8	3328	195.92.228.141	SOA in .theplanet.net
9	2454	10.16.1.40	Private range address
10	1963	66.28.225.156	SOA in .dns.cogentco.com

	# Alerts	Internal source
1	608566	130.85.5.89
2	371371	130.85.60.43
3	48530	130.85.6.49
4	45727	130.85.6.45
5	41365	130.85.28.2
6	39166	130.85.6.52
7	37698	130.85.253.10
8	31912	130.85.6.51
9	31395	130.85.6.50
10	26198	130.85.88.162

Table I - Top 10 Scan talkers - Internal destination and External destination

	# Alerts	Source - IP
1	82738	130.85.88.245
2	55504	130.85.1.3
3	41411	130.85.151.90
4	30567	130.85.1.4
5	29040	130.85.11.7
6	23720	130.85.6.45
7	23703	130.85.11.6
8	19567	130.85.60.43
9	15971	130.85.5.55
10	15654	130.85.6.60

	# Alerts	Destination - IP	Reverse dns-lookup
1	5767	216.40.225.68	h-216-40-225-68.23i.net
2	4345	216.239.51.101	www.google.com
3	4268	224.0.1.22	Multicast address (srvloc.mcast.net)
4	4228	131.118.254.39	a131-118-254-39.deploy.akamaitechnologies.com
5	3698	205.188.228.65	mslb4.spinner.com
6	3583	131.118.254.38	a131-118-254-38.deploy.akamaitechnologies.com
7	3223	205.188.228.129	mslb6.streamops.aol.com
8	3142	205.188.228.33	mslb3.spinner.com
9	3051	66.28.132.168	unf.unf.unf.u.nf
10	2959	66.62.70.248	SOA in .in-tch.com

Registration Information

IP **64.4.124.151** was chosen because of the targeted scanning to the internal host 130.85.88.165.

WHOIS information:

Ntelos North Cisco DSL DHCP Range #2 CFW-64-4-124-NNC (NET-64-4-124-0-1)
64.4.124.0 - 64.4.124.255
CFW Communications CFW-BLK-2 (NET-64-4-96-0-1) 64.4.96.0 - 64.4.127.255

Reverse DNS-lookup:

Host-name:	64-4-124-151.dmt.ntelos.net.
Nameservers:	ns.cfw.com and ns4.cfw.com.

Additional registration information:

OrgName:	CFW Communications
OrgID:	CFW
NetRange:	64.4.96.0 - 64.4.127.255
CIDR:	64.4.96.0/19
NetName:	CFW-BLK-2
NetHandle:	NET-64-4-96-0-1
Parent:	NET-64-0-0-0-0
NetType:	Direct Allocation
NameServer:	NS.CFW.COM
NameServer:	NS2.NTELOS.NET
NameServer:	NS1.NTELOS.NET
Comment:	ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:	2001-03-20
Updated:	2001-07-24
TechHandle:	DNS56-ORG-ARIN
TechName:	Domain Name Services
TechPhone:	+1-540-946-2638
TechEmail:	dns@cfw.com

CFW Communications (CFW)
Wilkes, Charles (CW284-ARIN) cfw@pobox.com +1-408-229-2748
CFW Network Inc. (AS7795) CFWNET 7795

Registered nets by CFW Communications which are traveled by a traceroute traversal:

CFW Communications CFW-BLK-1 (NET-216-12-0-0-1) 216.12.0.0 - 216.12.127.255
CFW Network - Router Sync Ports CFW-216-12-2 (NET-216-12-2-0-1) 216.12.2.0 - 216.12.2.255
CFW Network - WV Backbone #2 CFW-216-12-96 (NET-216-12-96-0-1) 216.12.96.0 - 216.12.96.255

IP **195.101.94.208** was chosen because this IP targeted the most *different* internal hosts wit OOS-packets and was also reported because of scanning to these hosts on port 80 and alerted as *Queso fingerprint.*

WHOIS information:

inetnum:	195.101.94.0 - 195.101.94.255
netname:	FR-ECHO
descr:	Socite ECHO
country:	FR
admin-c:	CR308-RIPE
tech-c:	CR308-RIPE
status:	ASSIGNED PA
notify:	addr-reg@rain.fr
mnt-by:	RAIN-TRANSPAC
changed:	noc@rain.fr 19970514
source:	RIPE
route:	195.101.94.0/24
descr:	ECHO
origin:	AS8891
mnt-by:	OLEANE-NOC
changed:	hostmaster@oleane.net 19980929
source:	RIPE
person:	Christophe RUELLE
address:	ECHO
address:	20 Parc des hautes Technologies

address:	06250 MOUGINS
phone:	+33 4 92 28 32 00
fax-no:	+33 4 92 28 32 01
e-mail:	ruelle@echo.fr
nic-hdl:	CR308-RIPE
notify:	addr-reg@rain.fr
mnt-by:	RAIN-TRANSPAC
changed:	nathalie@rain.fr 20000324
source:	RIPE

Reverse DNS-lookup:

Host-name:	x1crawler1-1-0.x-echo.com.
Nameservers:	ns.x-echo.com. ns1.x-echo.com.

Additional information:
This IP is reported at http://www.dshield.org/ipinfo.php?ip=195.101.94.208 and fightback was sent to ruelle@echo.fr on 2002-04-23 18:24:27. There is no reply received.

IP **12.151.57.37** was chosen because of the overwhelming scans originated from this host.

WHOIS information:

AT&T WorldNet Services ATT (NET-12-0-0-0-1)	12.0.0.0 - 12.255.255.255
ATLIGHTSPEED A-LIGHT112-56 (NET-12-151-56-0-1)	12.151.56.0 - 12.151.63.255

Reverse DNS-lookup:

Host-name:	no PTR-record
Nameservers:	ns1.securitas.bz. NS3.securitas.bz.ns.cfw.co

Additional registration information:

OrgName:	CFW Communications
OrgID:	CFW
NetRange:	64.4.96.0 - 64.4.127.255
CIDR:	64.4.96.0/19
NetName:	CFW-BLK-2
NetHandle:	NET-64-4-96-0-1
OrgName:	ATLIGHTSPEED
OrgID:	LSPD
NetRange:	12.151.56.0 - 12.151.63.255
CIDR:	12.151.56.0/21
NetName:	A-LIGHT112-56
NetHandle:	NET-12-151-56-0-1
Parent:	NET-12-0-0-0-1
NetType:	Reallocated
Comment:	
RegDate:	2001-10-11
Updated:	2002-08-22
TechHandle:	JM2923-ARIN
TechName:	McCoy, Jeff
TechPhone:	+1-720-264-2029
TechEmail:	jmccoy@atlightspeed.com

Additional information:
This IP is reported at http://www.dshield.org/ipinfo.php?ip=12.151.57.37 and there was no fightback sent. The attacked ports registered are 1350 and 1443.

IP **195.92.228.141**was chosen because this host was in the *Top Talkers list - scans* and all scans from this host are targeted at only 1 internal host as shown in the top 10 of internal hosts that were scanned by one external host. The targeted host is 130.85.151.95.

WHOIS information:

inetnum:	195.92.224.0 - 195.92.231.255
netname:	E2-WEB1
descr:	Energis Squared Managed Web Server Network
descr:	In case of problems, please contact +44 113 2346068
descr:	Please do not send abuse reports to tech or admin contacts
descr:	Abuse reports to abuse@energis-squared.com please!
country:	GB
admin-c:	PJ3130-RIPE
tech-c:	PJ3130-RIPE
rev-srv:	earth.theplanet.net
rev-srv:	venus.theplanet.net
rev-srv:	pluto.theplanet.net
status:	ASSIGNED PA
notify:	ripe-adm@planet.net.uk
mnt-by:	AS5388-MNT
changed:	darrenh@energis-squared.net 20001123
source:	RIPE
route:	195.92.0.0/16
descr:	Planet Online Limited
descr:	The White House
descr:	Melbourne St.
descr:	Leeds LS2 7PS United Kingdom
origin:	AS5388
mnt-by:	AS5388-MNT
changed:	matthew@planet.net.uk 19960612
source:	RIPE
person:	Pedro Jones
address:	Energis Squared
address:	Melbourne St
address:	Leeds, LS2 7PS
phone:	+44 113 207 6000
fax-no:	+44 113 2345656
e-mail:	pedro.jones@energis-squared.com
nic-hdl:	PJ3130-RIPE
mnt-by:	AS5388-MNT
changed:	ripe-adm@planet.net.uk 20010920
source:	RIPE

Reverse DNS-lookup:

Host-name:	no PTR-record
Nameservers:	earth.theplanet.net. pluto.theplanet.net. venus.theplanet.net.

IP **24.112.58.210** was chosen because this host was second in the *Top Talkers list - OOS* and was responsible for two alert types (*Null* scan and *INFO inbound GNUTella Connect* request) and targeted only internal host 130.85.150.209

WHOIS information:

OrgName:	Rogers Cable Inc.
OrgID:	ROCB
NetRange:	24.112.0.0 - 24.112.255.255
CIDR:	24.112.0.0/16
NetName:	ROGERS-CAB-1
NetHandle:	NET-24-112-0-0-1

Parent:	NET-24-0-0-0-0
NetType:	Direct Allocation
NameServer:	NS1.WLFDLE.RNC.NET.CABLE.ROGERS.COM
NameServer:	NS2.WLFDLE.RNC.NET.CABLE.ROGERS.COM
NameServer:	NS1.YM.RNC.NET.CABLE.ROGERS.COM
NameServer:	NS2.YM.RNC.NET.CABLE.ROGERS.COM
Comment:	
RegDate:	
Updated:	2002-08-20
TechHandle:	AD30-ARIN
TechName:	Budd, Paul
TechPhone:	+1-416-935-4729
TechEmail:	abuse@rogers.com

Reverse DNS-lookup:

Host-name:	CPE0000863e27fd.cpe.net.cable.rogers.com.	
Nameservers:	ns1.wlfdle.rnc.net.cable.rogers.com.	ns2.wlfdle.rnc.net.cable.rogers.com.
	ns1.ym.rnc.net.cable.rogers.com.	ns2.ym.rnc.net.cable.rogers.com.

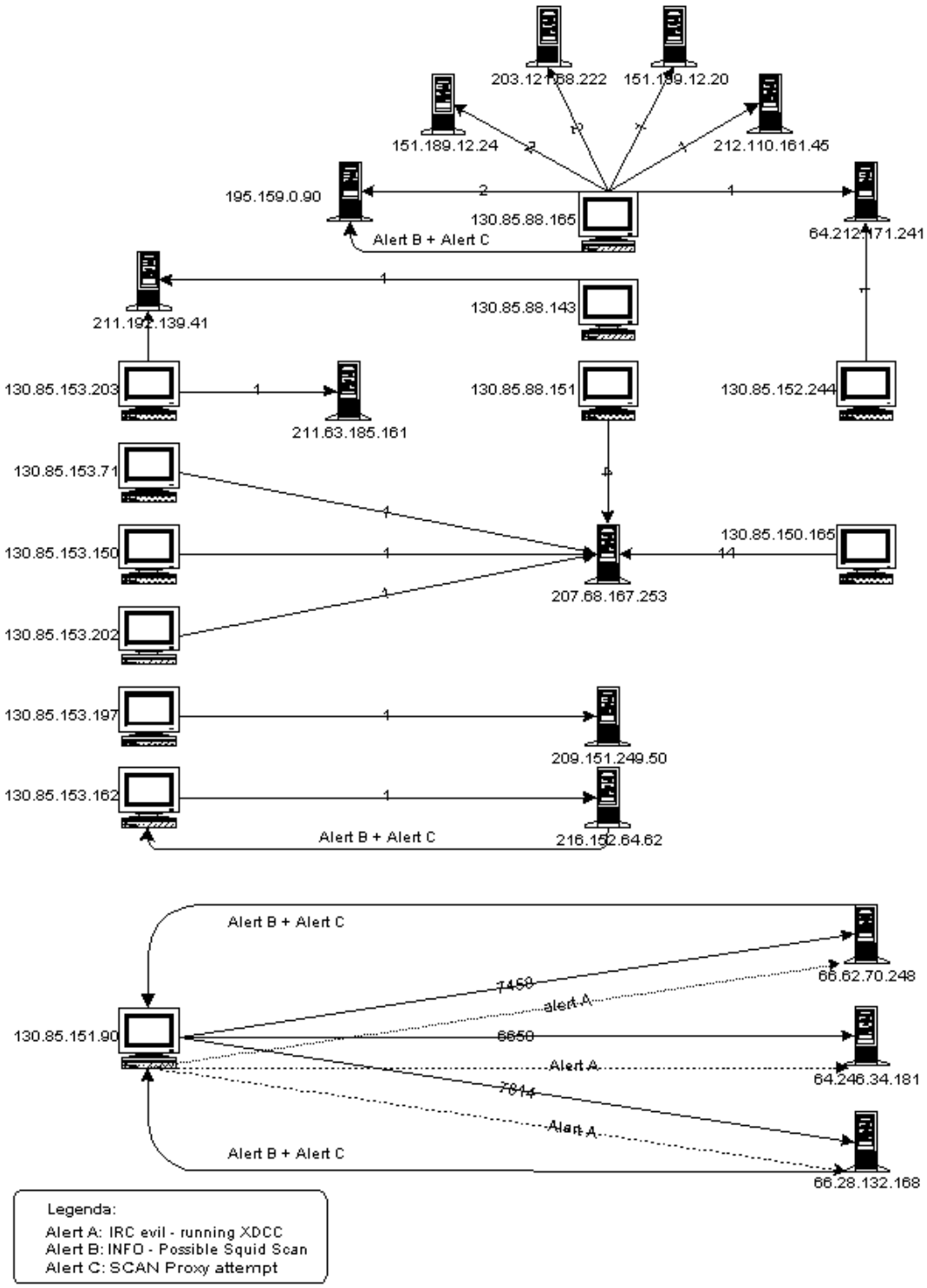
Link Graph

The link graph on the next page shows the connections that are detected due to the alert "*INFO Possible IRC Access*" and the correlated alerts that did arise for these connections. The local systems are grouped together according the subnet they reside on.

The graph also shows the traffic that is generated from potential IRC servers to the local hosts to test if the IRC-client is not behind some kind of proxy server. The number on the line, show the number of generated "*INFO Possible IRC Access*"-alerts. The two tables that are used for this graph can be found in paragraph "INFO Possible IRC Access ".

The local hosts are represented by a "monitor"-icon and the external hosts are represented by a "PC-tower"-icon. The local host 130.85.151.90 is a high volume user.

Link Graph


203.121.68.222
151.189.12.20
151.189.12.24
212.110.161.45
195.159.0.90
130.85.88.165
Alert B + Alert C
64.212.171.241
211.192.139.41
130.85.88.143
130.85.153.203
211.63.185.161
130.85.88.151
130.85.152.244
130.85.153.71
130.85.153.150
130.85.150.165
207.68.167.253
130.85.153.202
130.85.153.197
209.151.249.50
130.85.153.162
Alert B + Alert C
216.152.64.62
Alert B + Alert C
66.62.70.248
7450
alert A
130.85.151.90
6650
Alert A
64.246.34.181
7014
Alert A
Alert B + Alert C
66.28.132.168
Legenda:
Alert A: IRC evil - running XDCC
Alert B: INFO - Possible Squid Scan
Alert C: SCAN Proxy attempt

Analyses Process

The analyses started with a health check of the files that were provided. The more confident the log-files are, the more accurate the analyses will be.
Each set of the three different kind of logfiles was concatenated into one large file for each set and checked (See *appendix B:commands01*).

The global health check was performed on these three concatenated files. It is possible that the logging process produces malformed loglines due speed limitations on high volume networks. A script which counts the number of characters for each line and displaying the shortest and longest line found, *could* detect malformed log entries. Malformed log lines can often be detected because they are extremely long or extremely large. *Script01* was used to check this (See *appendix B:script01*). The same script was used to run against the independent files in each set of files for completeness.

A sample of the usage and output is given below:

```
cat work/alerts | ./script01
 Shortest line is:    77 characters
 06/11-10:15:53.593335  [**] Null scan! [**] 80.144.59.96:0 -> 130.85.88.179:0
 Longest  line is:    140 characters
 06/13-10:20:10.432574  [**] spp_portscan: portscan status from 130.85.28.2:
 3562 connections across 1 hosts: TCP(3561), UDP(1) STEALTH [**]
 Number of lines is:  780438 lines
```

It turned out that the files 'oos_Jun.11.2002.new' and 'oos_Jun.14.2002.new' both had one entry which was suspicious and indicated a malformed entry.

The script was also run against a subset of the 'work/alerts' file, excluding the 'spp_portscan' alerts: `cat work/alerts | grep -v 'spp_portscan' | ./script01`

Because there is a separate scan-file, the work/alerts file was split into two files:

```
grep -iv 'spp_portscan' work/alerts > work/alerts-no-scan
grep -i 'spp_portscan' work/alerts > work/alerts-only-scan
```

The *alerts-no-scan* file was processed by *Script02* to gather the most interesting alerts (See *appendix B:script02*). The script checks the alerts-file on three existing columns which are separated by '*[**]*'. The script can produce output for each column given on the command-line, separated by a '#'.

Perlscript *apr* was written to do various ways of research on the alert files and gathers information based on the source IP-address. The script can easily be altered to run against the scan files (example included in the source) or to gather the information based on the destination IP-address.

This perlscript and various *UNIX* commands like *sort, uniq, grep, sed, awk* and *vi* are used to gather all kind of information. Tools like *SnortSnarf, SnortRep* and *ACID* have not been used since it was fun digging into the logs without any prejudice.

References

1. Morris, Chris. “What Do You Do After You Deploy the IDS?”. January 3, 2001 URL: http://www.sans.org/newlook/resources/IDFAQ/deploy.htm
2. Staniford, Stuart. Paxson, Vern. Weaver, Nicholas. “How to 0wn the Internet in Your Spare Time” URL: http://www.icir.org/vern/papers/cdc-usenix-sec02/index.html
3. Internet Security Systems, Inc. “Executive Summary Internet Risk Summary”, for March 26, 2002 through June 24 2002 URL: https://gtoc.iss.net/documents/summaryreport.pdf
4. Staniford, Stuart. Hoagland, James. McAlerney, Joseph. “Practical Automated Detection of Stealthy Portscans” URL: http://www.silicondefense.com/pptntext/Spice-JCS.pdf
5. Provos, Niels. “Honeyd - Network Rhapsody for You”. URL: http://www.citi.umich.edu/u/provos/honeyd/
6. Liston, Tom. “Welcome To My Tarpit - The Tactical and Strategic Use of LaBrea” URL: http://www.hackbusters.net/LaBrea/LaBrea.txt
7. Schlotter, Chadd. “Anti-Hacking: The Protection of Computers”. April 2, 2001 URL: http://rr.sans.org/attack/antihack.php
8. Haig, Leigh. “LaBrea - A New Approach to Securing Our Networks”. Match 7, 2002. URL: http://rr.sans.org/attack/labrea.php
9. Distributed Intrusion Detection System. URL: http://www.dshield.org/
10. Internet Storm Center. URL: http://isc.incidents.org/
11. Security Focus DeepSight Analyzer (ARIS). URL: http://aris.securityfocus.com/
12. HackerWatch.org (McAfee Personal Firewall). URL: http://www.hackerwatch.org/
13. Provos, Niels. “honeyd creates network schizophrenia”. April 8, 2002. URL: http://archives.neohapsis.com/archives/sf/honeypots/2002-q2/0010.html
14. Homepage of the *LaBrea Tarpit.* URL: http://www.hackbusters.net/LaBrea/
15. Braun, Joakim von. “What port numbers do well-known trojan horses use?”. July 2002. URL: http://www.sans.org/newlook/resources/IDFAQ/oddports.htm
16. Stevens W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Longman Inc.1994 (17th printing April 2000).
17. Northcutt, Stephan and Novak, Judy and McLachlan, Donald. Network Intrusion Detection – An Analyst’s handbook. 2nd Edition New Riders Publishing. 2001
18. SamSpade, Information about registrations. URL: www.samspade.org.
19. Info on Error32. URL: http://www.dark-e.com/archive/trojans/err32/beta/
20. Error32. URL: http://www.megasecurity.org/trojans/e/error32/Error32.html
21. RadWare. “LinkProof” URL: http://www.radware.com/content/products/link.asp.
22. Romanski, James. “Using SNMP for Reconnaissance” August 12, 2000. URL:http://www.sans.org/newlook/resources/IDFAQ/SNMP.htm
23. MIB - Information. URL: http://www.mibcentral.com/index.shtml
24. SecurityFocus. “IPFilter TTL Fingerprinting Vulnerability” Updated April 2, 2002. URL:http://online.securityfocus.com/bid/4403/discussion/
25. Network Working Group. “SNMPv1”. May, 1990. URL:http://www.ietf.org/rfc/rfc1157.txt
26. Sans Institute. “How To Eliminate The Ten Most Critical Internet Security Threats The Experts’ Consensus Version 1.33” June 25, 2001. URL:http://www.sans.org/topten.htm

27. Chen, Yen-Ming . “Survey of Log Analysis Tools for Snort” July 2001. URL:http://www.unixreview.com/documents/s=1233/urm0107f/0107f.htm
28. Bryce, Alexander. “Port 137 Scan” May 10, 2000. URL:http://www.sans.org/newlook/resources/IDFAQ/port_137.htm
29. Incidents Org. “React”. URL: http://www.incidents.org/react/
30. Incidents Org. “Portinfo”. URL: http://isc.incidents.org/port_details.html?port=xxxx
31. Dshield Org. “IP-info”. URL: http://www.dshield.org/ipinfo.php?ip=aa.bb.cc.dd
32. So, Hee. Practical #0489. “GIAC Intrusion Detection In Depth - v3.0” February 16, 2002 URL: http://www.giac.org/practical/Hee_So_GCIA.doc
33. Beardsley, A. Tod. Practical #525 “Intrusion Detection And Analysis: Theory, Techniques, and Tools” May 8, 2002. URL:http://www.giac.org/practical/Tod_Beardsley_GCIA.doc
34. Fiddler, Matthew. Practical #484 “GIAC Intrusion Detection In Depth - v3.0” http://www.giac.org/practical/Matthew_Fiddler_GCIA.doc
35. Smith, Gary. Practical #0532. “GCIA Intrusion Detection In Depth - v3.1” 2002, URL: http://www.giac.org/practical/Gary_Smith_GCIA.zip

Appendix A - Overview of all alerts -

Alerts Total	*Rank*	*Alerts Local (L)*	*Rank*	*Alerts Out-bound (O)*	*Rank*	*Alerts In-bound (I)*	*Description of the Alert*
59224	1	59224					SMB Name Wildcard
52002	2	52002					SNMP public access
46878	7	383	1	43746	4	2749	Spp_http_decode: IIS Unicode attack detected
28111					1	28111	MISC Large UDP Packet
27142	3	27141	20	1			ICMP Echo Request L3retriever Ping
21959			2	21959			INFO Possible IRC Access
8471			3	4154	3	4317	INFO MSN IM Chat data
4906					2	4906	AFS - Off-campus activity
3623	4	3623					ICMP Echo Request Nmap or HPING2
3538	5	2973			10	565	High port 65535 udp - possible Red Worm - traffic
2419	22	1	4	2418			spp_http_decode: CGI Null Byte attack detected
2333					5	2333	WEB-MISC Attempt to execute cmd
2179					6	2179	FTP DoS ftpd globbing
2144					7	2144	Watchlist 000220 IL-ISDNNET-990517
1268	12	13	5	1255			ICMP Fragment Reassembly Time Exceeded
909			6	909			ICMP Router Selection
752					8	752	INFO Inbound GNUTella Connect request
620					9	620	WEB-IIS view source via translate header
528			7	528			INFO Outbound GNUTella Connect request
490			8	468	22	22	Incomplete Packet Fragments Discarded
401	6	401					ICMP Destination Unreachable (Communication Admin. Prohibited)
317	28	1			11	316	Null scan!
260					12	260	IDS552/web-iis_IIS ISAPI Overflow ida nosize
212	8	71	9	141			ICMP Echo Request Windows
209					13	209	SCAN Proxy attempt
95			10	95			ICMP Echo Request CyberKit 2.2 Windows
87					14	87	WEB-IIS _vti_inf access
80					15	80	WEB-FRONTPAGE _vti_rpc access
79			11	79			IRC evil - running XDCC
79					16	79	INFO - Possible Squid Scan
71					17	71	INFO FTP anonymous FTP
65	9	55	18	4	36	6	Possible trojan server activity
53					18	53	WEB-MISC http directory traversal
45			12	45			WEB-MISC 403 Forbidden
39					19	39	WEB-IIS Unicode2.pl script (File permission canonicalization
38			13	38			INFO Napster Client Data
33	10	33					ICMP traceroute
32	11	32					ICMP Destination Unreachable (Protocol Unreachable)

Alerts Total	*Rank*	*Alerts Local (L)*	*Rank*	*Alerts Out-bound (O)*	*Rank*	*Alerts In-bound (I)*	*Description of the Alert*
32	20	2			21	30	NMAP TCP ping!
31					20	31	Watchlist 000222 NET-NCFC
28			14	28			WEB-IIS Unauthorized IP Access Attempt
27	14	5	15	22			ICMP Echo Request Delphi-Piette Windows
21					23	21	WEB-CGI scriptalias access
20	19	3	17	7	32	10	High port 65535 tcp - possible Red Worm - traffic
20					24	20	WEB-MISC compaq nsight directory traversal
19					25	19	UDP SRC and DST outside network
14					26	14	EXPLOIT x86 setuid 0
14					27	14	EXPLOIT x86 NOOP
12	13	8			38	4	Back Orifice
11					28	11	SCAN Synscan Portscan ID 19104
11					29	11	SCAN FIN
10					30	10	suspicious host traffic
10					31	10	Queso fingerprint
8	15	5			40	3	Attempted Sun RPC high port access
8	26	1			35	7	SUNRPC highport access!
8			16	8			IFO Inbound GNUTella Connect accept
8					33	8	EXPLOIT x86 setgid 0
8					34	8	EXPLOIT NTPDX buffer overflow
5					37	5	WEB-CGI redirect access
4	16	4					ICMP Destination Unreachable (Host Unreachable)
4	17	3			47	1	Port 55850 udp -Possible myserver activity-ref. 010313-1
3	18	3					Port 55850 tcp -Possible myserver activity- ref. 010313-1
3					39	3	MISC PCAnywhere Startup
2	21	2					ICMP Echo Request BSDtype
2	23	1			43	1	Virus - Possible scr Worm
1	24	1					Virus - Possible pif Worm
1	25	1					Virus - Possible MyRomeo Worm
1	27	1					Probable NMAP fingerprint attempt
1			19	1			NIMDA - Attempt to execute cmd from campus host
1					41	1	X11 outgoing
1					42	1	WEB-CGI formmail access
1					44	1	TFTP - Internal UDP connection to external tftp server
1					45	1	TFTP - External UDP connection to internal tftp server
1					46	1	SCAN XMAS
1					48	1	MISC traceroute
1					49	1	EXPLOIT x86 stealth noop

Appendix B - Commands and scripts used for assignment 3 -

Commands01

```
# Create for all three sets, one concatenated file.
cd $HOME/logs     ## Here are all the logfiles gathered.
######### Processing of the oos_* files ############################
# Get rid of the '^M' character and empty lines in the oos_* files:
# contents of cmd01:
#             s/^M//
#             /^$/d
#
sed -f cmd01 oos_Jun.11.2002 > oos_Jun.11.2002.new
sed -f cmd01 oos_Jun.12.2002 > oos_Jun.12.2002.new
sed -f cmd01 oos_Jun.13.2002 > oos_Jun.13.2002.new
sed -f cmd01 oos_Jun.14.2002 > oos_Jun.14.2002.new
sed -f cmd01 oos_Jun.15.2002 > oos_Jun.15.2002.new
# Open the *.new files in vi to check the eof-character.
# Concatenate all the oos_* files into one file:
cat oos*.new > work/oos
wc oos*new   57     491    3334 oos_Jun.11.2002.new
              5      27     256 oos_Jun.12.2002.new
             65     384    3471 oos_Jun.13.2002.new
             45     342    2642 oos_Jun.14.2002.new
             15      90     820 oos_Jun.15.2002.new
                                187    1334   10523 total
wc work/oos    ----> Gives:      187    1334   10523
######### Processing of the scan.* files ############################
cat scans* >work/scans
wc scans*    389857 2786572 24516007 scans.020611
             288892 2057369 18028165 scans.020612
             426284 3092233 27031631 scans.020613
             420811 3036813 26575222 scans.020614
             282283 2004375 17422461 scans.020615
             321829 2288419 19927071 scans.020616
                                2129956 15265781 133500557 total
wc work/scans  ----> Gives:      2129956 15265781 133500557
######### Processing of the alert.* files ############################
cat alert* > work/alerts
%wc alert*   141476 1933377 17253046 alert.020611
             108300 1443462 12937967 alert.020612
             167161 2154342 19537545 alert.020613
             164805 2137948 19349046 alert.020614
              88022 1177917 10439117 alert.020615
             110674 1434887 12851421 alert.020616
                                780438 10281933 92368142 total
wc work/alerts ----> Gives:      780438 10281933 92368142
######################################################################
# Checking the concatenated files:
cat work/alerts | ./script01
cat work/oos | ./script01
cat work/scans | ./script01
cat oos | grep '\->' | ./script01
grep -iv 'spp_portscan' work/alerts | ./script01
grep  '[^^]06\/1' alerts ## check if there is more than 1 entry in
                         ## one line.
######################################################################
# Extracting the portscan entries out of the alerts file:
grep -iv 'spp_portscan' work/alerts > work/alerts-no-scan
grep -i 'spp_portscan' work/alerts > work/alerts-only-scan
```

Script01

```
#!/bin/sh
#
# Analyze data on default input:
#      Number of characters in shortest line on input
#      text of this shortest line in input
#      Number of characters in longest line on input
#      text of this longest line in input
#      Total number of lines on input
#
awk '
BEGIN {
 min = 65535 ; max = 0;  tot =0
}
{
# Determine the longest line
if (length($0) > max ) {
   max = length($0) ;
   lline = $0 ;}
# Determine the shortest line
if (length($0) < min ) {
   min = length($0) ;
   sline = $0 ;}
tot++ ;
}
END {
 print "Shortest line is:   ", min , "characters" ;
 print "\t"sline;
 print "Longest  line is:   ", max , "characters";
 print "\t"lline;
 print "Number of lines is: ", tot , "lines";
}'
```

Script02

```
#!/bin/sh
# Analyze Alert files:
#      Total number of lines
#      Check min and max number of fields (should always be 3)
#      - spp_portscan has a trailing ' ' --> 3 fields
#      when a number is given, that field is printed
#      otherwise the complete line is printed on stdout
#
#  sample:  script02 1 3 < input prints columns 1 and 3,
#           seperated by '#'
#
if [ $1 ]
  then  fld=\$$1;
  if [ $2 ]
     then fld=\$$1\"#\"\$$2 ;
       if [ $3 ]
          then fld=\$$1\"#\"\$$2\"#\"\$$3 ;
       fi
  fi
else
  fld='$0' ;
fi
awk  '
BEGIN { FS="[ ]+\\[\\*\\*\\][ ]+"  ;
 min = 65535 ; max = 0;  tot =0 ;
}
{
# Determine the number of fields (should always be 3)
if (NF  > max )
```

```
   max = NF ;
if (NF < min )
   min = NF ;
#print $1"+"$2"+"$3 ;
 if ($fld != "") print '$fld';
tot++ ;
}
END {
 if (min != 3) print "ERROR -- There is a line with only " min " field(s)";
 if (max != 3) print "ERROR -- There is a line with " max " fields" ;
# else if (fld) print "$fld"; else print $1, $2, $3 ;
 print "Number of lines processed is: ", tot ;
# print "max is:"max   "min is"min ;
}' -
```

Perlscript apr - alert port reporter

```
#!/usr/bin/perl -w
#
# Program apr -- Alert Port reporter
#
# Input: Snort alert-log (Fast-mode)
#        only alerts (no spp_portscan)
#
# Assumed lay-out:
# MM/DD-HH:MM:SS.999999  [**] --- Description --- [**] AAA.BBB.CCC.DDD:PPPPP
# -> EEE.FFF.GGG.HHH:PPPPP
#
# Purpose: Define filter(s) and report the port usage and destinations
#          Analyze only udp and tcp connections
#
#
use strict 'vars';       # Declare all variables
use Getopt::Std;
#
#
my %opts;
my %srcip;
#
my $dq='[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' ; # Dotted-quad repr.
my $pt='\d{1,5}' ;                                  # port repr.
#
my $df=$dq ;                                        # default destiniation filter
my $sf=$dq ;                                        # default source filter
#
if ( ! getopts("hs:d:nta", \%opts)) {            # unknown option
print "run\n";
print "options %opts\n";
   &printhelp;
   exit 1;
}
#
if ($opts{h}) {                                     # Help requested
   &printhelp;
   exit 0;
}
#
if ($opts{s}) {                                     # source filter specified
   $sf = $opts{s} ;
   print "Source filter used is:      $sf\n" if (not $opts{t}) ;
}
if ($opts{d}) {                                     # destination filter specified
   $df = $opts{d} ;
   print "Destination filter used is: $df\n" if (not $opts{t}) ;
```

```
}
#
if ($#ARGV+1 <1) {
   &printhelp;
   exit 1;
}
#
my $infile = $ARGV[0];
#
main();
#
sub main {
  open ( I, $infile) || die "*** can not open $infile $!";
#
  while (<I>) {
    # Fill all the hashes ....
    /(^.*\b)\s+\[\*\*\]\s+(.*)\s+\[\*\*\]\s+($sf):($pt) -> ($df):($pt)/;
##or this one for scan files## /(^.*\b)\s+(.*)\s+($sf):($pt) -> $df):($pt).*/;
    # $1 = Time/Date   $2 = Description        $3 = Src-ip
    # $4 = srcport     $5 = dest-ip            $6 = dstport
    next if (!defined $6) ;
    if ($srcip{$3}) {                              # src-ip already processed
        if ($srcip{$3}->{"$5"}) {                  # and with this dest-ip
            $srcip{$3}->{"$5"}->[0]++ ;            # Total connects ;
            if ($srcip{$3}->{"$5"}->[2]->{"$6"}) {   # add 1 to the port
               $srcip{$3}->{"$5"}->[2]->{"$6"}++
            }
            else {
               $srcip{$3}->{"$5"}->[1]++ ;         # add to # of ports
               $srcip{$3}->{"$5"}->[2]->{"$6"} = 1 ;
            }
        }
        else {  # create entry for dst-ip to this src ip
            my $rdp = {}   ;  # Create empty referenced hash for the dstport
            $rdp->{"$6"}=1 ;
            my $a =[]      ;  # create empty referenced array
            $a->[0] = 1    ;  # connects to this IP
            $a->[1] = 1    ;  # Count the different ports
            $a->[2] = $rdp ;  # Placeholder for the ports
            $srcip{$3}->{"$5"}=$a ;
        }
    }
    else {                    # create a new entry for this source ip
      my $rdp = {}   ;        # Create empty referenced hash for the dstport
      $rdp->{"$6"}=1 ;
      my $a = []     ;        # create empty referenced array
      $a->[0] = 1    ;        # connects to this IP
      $a->[1] = 1    ;        # Count the different ports
      $a->[2] = $rdp ;        # Placeholder for the ports
      my $rd = {}    ;        # Create empty referenced hash for the dst ip's
      $rd->{"$5"}=$a ;        # assign the array withinfo
      $srcip{$3}=$rd ;        # add this src-ip to the hash
    }
  } #eow
  close I ;
  &printheader() if (not $opts{t});
  &printsourceip() ;
}
#
sub printhelp {
   print "Usage: apr [-h][-n][-t][-s 'IP'][-d 'IP']  <filename> \n\n";
   print "\t<filename> is te name of the snort portscan log-file\n\n";
   print "\tFlags:\n";
   print "\t-h\tshow this help message\n";
```

```
   print "\t-a\tDo not display details\n";
   print "\t-n\tDisplay totals for each destination-ip\n";
   print "\t-t\tDisplay the totals for each destination-ip\n\n";
   print "\t-s\t'IP' report on source-ip only\n";
   print "\t-d\t'IP' report on destination-ip only\n";
   print "\t\t'IP' can be a perl regular expression\n" ;
}
#
sub printheader {
  print "\tOverview of file $infile:\n" ;
  print "\t=========================\n\n" ;
}
#
sub printsourceip {
  my $t0 = 0 ;                      # Number of processed source-ip's
  my $t1 = 0 ;                      # Number of destination ip's
  my $t2 = 0 ;                      # Number of different port
  my $t3 = 0 ;                      # Number of probes
  # Sorted on Source IP
  my @sortkeys = sort  {ip2fullip($a) cmp ip2fullip($b)} keys %srcip ;
  foreach (@sortkeys) {
    $t0++ ;
    my $c1 = $_ ;                                # Column 1
    my $r = $srcip{$_}  ;
    my @sortkeys2 = sort keys %{$r} ;
    $t1 = 0 ;                                   # Count the number of source-ip's
    $t2 = 0 ;                                   # Number of ports
    $t3 = 0 ;                                   # Number of probes
    my %prtcnt    ;                             # Create empty named hash
    foreach (@sortkeys2) {
      $t1++;                                    # Count the destination ip's
      my $c2 = $_ ;                             # Column 2
      my $rp = $r->{$_}->[2] ;                  # Reference to the port hash
      my @sortkeys3 = sort keys %{$rp};
      foreach (@sortkeys3) {
        my $p=$_ ;
        if (not $opts{a}) {
            print "From $c1 to $c2 on port $p ($rp->{$_} times)\n";
        }
        if (not exists($prtcnt{"$p"})) {
           $prtcnt{"$p"}=0 ;
           $t2++ ;
        }
        $t3 = $t3 + $rp->{$_} ;
      }
      if ($opts{t}) {                         # Subtotals per dst-ip
         print "     $c1 to $c2 are $r->{$_}->[0] connections on" ;
         print " $r->{$_}->[1] port(s)\n";
      }
    }
    if ($opts{n}) {                           # Subtotals per source-ip
       print "IP   $c1 probed $t1 destination(s), $t2 distinct port(s) ($t3
alerts)\n";
    }
  }
  print "\n";
}
#
sub ip2fullip() {                             # Needed for sorting the source-ip
  my ($ip) = @_ ; my @a = split (/\./, $ip)  ;
  for (my $i = 0; $i < 4 ; $i++ ) {
      while (length($a[$i]) < 3) { $a[$i] = '0'.$a[$i] ;}
  }
  my $fullip = join('.', @a) ; return $fullip ;
```