GIAC Certified Intrusion Analyst (GCIA) Practical Version 3.3

by Daniel Wesemann

24 March 2003

SANS Network Security 2002, Washington D.C

Table of Contents

Assignment Part #1:
Describe the State of Intrusion Detection

Getting the most out of Cisco PIX Firewall logs

Introduction

The installation and configuration of a Cisco PIX firewall has been covered in other papers, like the one by Steve Textor [1]. In addition to the information contained therein, this paper is intended to guide the reader through some of the configuration steps needed to get a Cisco PIX firewall to provide meaningful logs, and also to offer some help in the interpretation and analysis of PIX firewall log messages.

Configuring a PIX for meaningful logging

How to configure a PIX firewall to forward its logs to a remote syslog server is covered by various Cisco documents like [2] and [4]. Basically everything you ever wanted to know about Syslog and syslog servers in general is covered on Tina Bird's Log Analysis web site [6]. Hence, I'm assuming for this paper that you got both, a running syslog server and a Cisco PIX firewall which is forwarding its logs to said server. More information on all the PIX firewall configuration commands used in this document can be found in the PIX Firewall Command Reference [8] available on the Cisco website.

Configure the Syslog Log Level

Unlike many other firewall products, PIX firewalls do not allow you to directly influence which events or access list entries actually generate log messages. The one thing you *can* specify is how "serious" an event must be in order to leave a trace in the logs. Log levels start at "emergencies (0)" and extend up to "debugging (7)", with increasingly verbose results in the logs. "Higher" (more verbose) log levels always also include all the messages generated by the lower levels. Rather than to risk a heated debate about the "correct" log level for a PIX firewall, let me simply state that my preferred level is "notifications (5)". Since the PIX log messages are listed ranked by severity on the Cisco documentation page [7], it is easy to verify which messages you are going to miss or additionally incur by lowering or raising the log level from "notifications (5)" to something else.

```
mypix# logging trap notifications
```

Configuring Log Timestamps

Since every respectable syslog server will stamp all incoming data with the current date and time anyway, you can sometimes do without the PIX adding a time stamp to the log messages on its own. While this will reduce the size of an average log entry (and consequently also the log storage space required) by about 20%, if you *really* intend to use the firewall logs for correlation purposes with IDS logs and possibly also for forensic analysis, you'll definitely need the PIX to log with accurate time stamps of its own.

```
mypix# ntp server <primary ntp server ip> source inside
mypix# ntp server <backup ntp server ip> source outside
mypix# logging timestamp
```

Enabling the rudimentary PIX Intrusion Detection features

There's in fact a small Network IDS built into every PIX firewall. The main purpose of this feature set seems to be to acquaint the user with Cisco IDS terminology and to eventually get him/her to buy a "proper" IDS from Cisco. Nevertheless, the feature set is there, and we might just as well make use of it. The two PIX IDS configuration entries which show up in every PIX configuration by default are

```
ip audit info action alarm
ip audit attack action alarm
```

which only define the default behaviour of the PIX should an "informational" or "attack" IDS signature trigger on a data packet. A default response of "alarm" means that the PIX will only log the event to the configured syslog server. More militant options include "drop" and "reset" as active responses, but I would not recommend them for production use.

Normally, I enable the "attack" and "informational" signatures on the outside interface of every PIX firewall.

```
mypix# ip audit name inbound-attack attack action alarm
mypix# ip audit interface outside inbound-attack
mypix# ip audit name inbound-info info action alarm
mypix# ip audit interface outside inbound-info
```

I'll touch on some of the syslog messages you can expect to get from the PIX IDS feature later in this document. If you find that the "informational" signatures generate too much clutter in your log, refer to the section of this paper dealing with "fine-tuning" of PIX log messages.

Enabling the "fragguard" feature

One more PIX feature which you should enable on an Internet-facing firewall is the fragguard option. This feature ensures that the PIX only passes packet fragments for connections for which the "initial" fragment has already been processed an verifyed against the access control lists. Fragguard also limits the rate to 100 fully reassembled fragmented packets per internal host and second. Because these two sensible security could impair connectivity through particularly busy PIX firewalls burdened with many fragmented packets, I suggest that you refer to the Cisco documentation [8] and try the setting out in a lab environment. You do *not* need to enable fragguard for the PIX to catch insidious fragmentation attacks like Teardrop with tiny or overlapping fragments - the PIX will catch and discard these by default. Personally, I've been running fragguard on various PIX 6.2 installations without encountering any adverse effects.

```
mypix# sysopt security fragguard
```

Making sense of the Cisco PIX log entries

While Cisco are doing a great job at documenting the PIX firewall log messages themselves [3], there is pityingly little research or documentation available on what attack, packet or event *causes* a particular message to appear in the logs. Watching the attack signatures in a PIX firewall log is sort of like monitoring an IDS where you do not have access to the IDS ruleset - you are forced to fly "by instruments" and have to trust whatever the system throws at you. While this problem is not of importance for the typical firewall log messages caused by denied connections, some of the more attack-centric log entries would be much more useful if Cisco would see it fit to properly document the possible causes leading to these events.

A very good analysis of syslog messages generated by a PIX when probed with certain crafted packets has been posted to the GIAC list by Curt Wilson back in November 2000. The paper [5] is still one of the few devoted to this subject and recommended reading for every Intrusion Analyst who has a PIX firewall somewhere in his/her jurisdiction.

Using the Severity Level as a first guidance

PIX log messages are, as mentioned earlier, grouped by level of severity. Alert messages (level 1) basically deal with hardware related events like interfaces losing connectivity or a firewall failing over to the secondary unit. Level 2 to level 5 messages deal mainly with things going wrong because of (potentially malicious) outside influence and also provide information on the PIX's state and health. Level 6 and 7 introduce verbose logging of connections, including those permitted by the ruleset, and also provide some information to track activity of some of the lesser used PIX features (user auth, built-in DHCP, etc).

If your PIX is configured with "logging trap notifications", as recommended earlier, messages of level 6 and 7 will never appear in your syslog.

Some PIX Log messages explained

`%PIX-2-108002: SMTP replaced chars`

This message can show up in the logs if you are sending or receiving SMTP email through your PIX firewall and you have the PIX SMTP security "fixup" feature turned on (which you should). Cisco documentation on the exact behaviour of "fixup smtp" is almost nonexistent, but I have captured sufficient messages of this type to assume that the PIX is monitoring the email address syntax used in the "To" and "From" fields of mail messages and jumps in by replacing the offending characters with spaces whenever the address does not comply with RFC 821. This mainly seems to defuse sneaky attempts to use the "pipe" symbol and attempts at file redirection using multiple occurences of ">". As far as I can tell, this feature should also catch attempts to exploit the recent Sendmail vulnerability (CVE CAN-2002-1337), but I haven't seen any evidence in the logs so far.

`%PIX-2-106020: Deny IP teardrop fragment`
Somebody is pestering your firewall with fragmented packets which overlap on reassembly. This so-called "teardrop" attack used to bring a certain OS to its knees a couple of years ago, but is not overly dangerous anymore. Nevertheless, overlapping packet fragments to not "naturally" thrive in the packet space, and can thus be safely classified as hostile activity. The PIX will detect and discard such attempts independent of whether you have the "fragguard" feature turned on or not.

`%PIX-3-305005: No translation group found`
Your PIX received a packet for a destination which is by virtue of routing "behind" the PIX, but the firewall does not have a corresponding "static" translation entry allowing an inbound connection. In other words, somebody from the outside is trying to access a system on your DMZ or inside network which either does not exist or to which you have decided not to provide inbound access. This error message is one of the neat features of PIX firewall. With the logs of many other firewall brands, it is impossible to distinguish form the log message whether the inbound access attempt was denied by the ruleset or headed for a non-existent system. You can safely assume that traffic destined for non-existing systems is either the result of a typo or misconfiguration, or it is hostile (reconnaissance) activity.

`%PIX-3-106011: Deny inbound (No xlate)`
Somebody tried to open an inbound connection through an address which is used for outbound port address translation (PAT). If the destination port used by the outside party is not currently in use as a "PAT" port of an inside-out connection, the PIX will log the attempt with this error message and send a reset packet back to the offending party. If the port *is* currently in use for an inside-out connection, the PIX will log the attempt as an ACL violation (%PIX-4-106023) and *not* send a reset packet back.

`%PIX-3-313001: Denied ICMP type x, code y`
This message appears in the logs if somebody is sending ICMP packets to the interface address of the firewall itself. ICMP packets trying to pass "through" the firewall will be matched against the ACLs like any other packet, but ICMP packets "terminating" on the firewall are being treated differently. By default, the PIX will accept all ICMP traffic on all its interfaces and thus this error message will never show up in the logs. Refer to the PIX command reference [8] on the "icmp" command for information on how to change the default behaviour.

`%PIX-4-106023: Deny TCP connection by access-group`

This is the standard run-of-the-milk log entry which signifies that your firewall is doing its job and rejecting packets which have no ACL entry permitting them. Note that this message only appears for inbound connection attempts if an address translation entry ("static") is present for the destination host; otherwise, the packet will get logged as %PIX-3-305005 (see above).

`%PIX-4-402106: Rec'd packet not an IPSec packet`
This message usually appears in the logs if somebody from the outside is trying to telnet into your PIX. A PIX only accepts encrypted Telnet management connections from the outside interface.

`%PIX-4-500004: Invalid transport field`
This somewhat misleading message appears if either the source or destination port of an UDP or TCP packet trying to pass through the PIX is 0 (zero). Since using a source port of zero is the default setting of the HPing2 scan tool, this message often shows up in the logs when an unsavvy user is running a HPing2 scan against your firewall or DMZ.

`%PIX-5-304001: Accessed URL`
When configured with a log level of "notifications", the PIX http fixup engine will also log every accessed URL to the syslog server. Since this information can also be found in webserver or proxy logs, I usually opt to disable this specific log message (see the next section on "Fine-Tuning PIX logs" for details)

`%PIX-5-111007: Begin configuration reading from terminal`
Somebody is about to reconfigure your PIX. In an ideally secure world, this would be somebody known to you :-)

`%PIX-5-500003: Bad TCP hdr length (hdrlen=28, pktlen=20)`
Somebody tries to send a packet through your PIX where the TCP header is either corrupt or crafted. The "hdrlen" value refers to the length of the TCP header as specified in the "offset" field of the TCP header, "pktlen" refers to the expected total length of the TCP portion (header plus payload) as derived from the IP header. "hdrlen" values smaller than the minimally required 20 bytes will trigger this log message, as will, for obvious reasons, packets where the header claims to be bigger than pktlen.

Some PIX IDS log messages explained

`%PIX-4-400014: IDS:2004 ICMP echo request`
This IDS message is pretty much useless - it will appear in the logs whenever somebody is trying to ping a device through the PIX. Since the IDS rule seems to "catch" the packet before it gets evaluated against the access lists, *every* ICMP ping arriving at your PIX will trigger this message. Other IDS messages of the IDS:200x range are equally annoying. But rather than to disable the entire IDS feature again, simply turn off these particular messages. Refer to the next section titled "Fine-tuning PIX logging" on how to achieve this.

`%PIX-4-400021: IDS:2011 ICMP Address Mask Request`

ICMP mask requests are sometimes used in the "intelligence gathering" phase of an attack to determine how good the firewalling/filtering is and whether a certain device can be reached with ICMP packets other than ping. You'll likely see this IDS message in conjunction with its peer IDS signatures, the "ICMP time request" and "ICMP info request".

`%PIX-4-400024: IDS:2151 Large ICMP Packet`
Since the documentation is pretty silent with respect to what Cisco deems to be a "large" ICMP packet, this message is of questionable value. The message appears rarely enough on my PIXes connected to the Internet to suggest that Cisco has set the limit reasonably high, somewhere beyond 1000 bytes.

`%PIX-4-400023: IDS:2150 ICMP Fragment`
This appears to be the sister message of the one above, and I've seen it appear in the logs only if somebody has apparently been cycling through various ping payload sizes (for whatever reason). If this happens, you'll first see a normal %PIX-4-106023 deny entry when the ACLs catch the unsuspiciously sized initial pings. Then the IDS feature will kick in and report the "Large ICMP Packets" once the ping size moves beyond 1000 bytes or so. Once the ping size becomes bigger than the path MTU, it will be split into fragments, which then will trigger this "ICMP Fragment" message.

`%PIX-4-400026: IDS:3040 TCP NULL flags`
This message is an indication that somebody is playing with the TCP header. No respectable TCP packet would allow itself to be seen without wearing any flags. A small familiy of PIX IDS messages is devoted to this and other odd combinations of TCP flags, with separate messages for NULL, SYN+FIN and FIN-only packets.

`%PIX-4-400008: IDS:1102 IP Land Attack`
Somebody is plying one of your servers with the ages-old "Land Attack", which consists of an IP packet where the source and destination IP address happen to be the same. This message is somewhat redundant, since it will be accompanied by %PIX-2-106017, "Deny IP due to Land Attack".

Fine-tuning the logs

Lowering the log level will steadily reduce your chance to come across some rare gem in the logs, to encounter some message which you've never seen before and which alerts to you something untoward happening on your network. Therefore, rather than lowering the log level, I suggest that you use the available configuration options to selectively disable those log messages which you percieve provide no value.
The most straight-forward approach to squelch annoying log messages is by simply disabling them, using the "no logging message <id>" command. For IDS messages, the prefered way is to turn off the signature itself rather than to suppress the resulting message. The two examples shown below serve to suppress the "Accessed URL" log message and to turn off the IDS signature triggering on ICMP echo requests.

```
mypix# no logging message 304001
mypix# ip audit signature 2004 disable
```

Automating the processing of PIX Logs

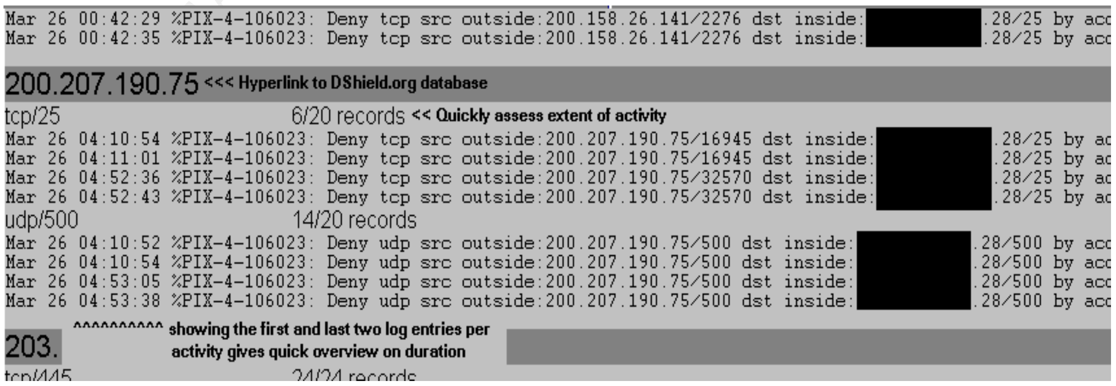
There's a number of tools and scripts available which help in the processing and evaluation of Cisco PIX firewall logs. The more you are willing to pay (ranging from nothing to a coupla hundred dollars), the more colorful your resulting packet statistics will be.
The more prominent freeware tools that I am aware of are

- pix2ss.pl, a Perl script which converts PIX logs into a format suitable for post-processing it with SnortSnarf.
- pix-summarize, a highly configurable Perl script written by Liudvikas Bukys from Rochester University.
- fwlogwatch, a very universal log parser written by Boris Wesslowski of University of Stuttgart

You might find others more by regularly checking on Tina Bird's Log Analysis web page [6]. Personally, I am using fwlogwatch, but I also find myself relying more and more on, yes, yet another Perl script which I have written on my own. Working with tools like the ones listed above, I found that I'm not particularly fond of just receiving *statistics* at the end of the day, but would rather browse through some of the real events. This is why I wrote "pixtract.pl".

What I consider to be one of the key features of "pixtract.pl" is that it actually lists the first and last two log entries of everything the script classifies as interesing activity. Your perception might vary, but I prefer to look at the actual log entries rather than at some translation, because my experience is that every translation step tends to falsify or obfuscate the actual information. Another feature of "pixtract.pl" is that it goes by the PIX syslog message numbers, and will list separately all messages which it does not know yet. This ensures that I'm not missing anything "new" by simply parsing for messages which I've seen before. Output is in HTML, with all IP addresses as hyperlinks pointing to the DShield database for quick cross referencing.

An excerpt from a log file parsed by "pixtract.pl" is shown below.



```
Mar 26 00:42:29 %PIX-4-106023: Deny tcp src outside:200.158.26.141/2276 dst inside:[redacted].28/25 by acc
Mar 26 00:42:35 %PIX-4-106023: Deny tcp src outside:200.158.26.141/2276 dst inside:.28/25 by acc

200.207.190.75 <<< Hyperlink to DShield.org database
tcp/25                          6/20 records << Quickly assess extent of activity
Mar 26 04:10:54 %PIX-4-106023: Deny tcp src outside:200.207.190.75/16945 dst inside:[redacted].28/25 by ac
Mar 26 04:11:01 %PIX-4-106023: Deny tcp src outside:200.207.190.75/16945 dst inside:.28/25 by ac
Mar 26 04:52:36 %PIX-4-106023: Deny tcp src outside:200.207.190.75/32570 dst inside:.28/25 by ac
Mar 26 04:52:43 %PIX-4-106023: Deny tcp src outside:200.207.190.75/32570 dst inside:.28/25 by ac
udp/500                         14/20 records
Mar 26 04:10:52 %PIX-4-106023: Deny udp src outside:200.207.190.75/500 dst inside:[redacted].28/500 by acc
Mar 26 04:10:54 %PIX-4-106023: Deny udp src outside:200.207.190.75/500 dst inside:.28/500 by acc
Mar 26 04:53:05 %PIX-4-106023: Deny udp src outside:200.207.190.75/500 dst inside:.28/500 by acc
Mar 26 04:53:38 %PIX-4-106023: Deny udp src outside:200.207.190.75/500 dst inside:.28/500 by acc
          ^^^^^^^^^^ showing the first and last two log entries per
203.                 activity gives quick overview on duration
tcp/445                         24/24 records
```

Since most of the script was written during work time paid by my employer, I cannot currently relase the script to the public. I am, though, working on a completely new version of the script, which I have implemented from scratch and on my own time. The result will be made available under GPL in due course.

References

[1] Textor, Steve: "The Installation and Configuration of a Cisco PIX Firewall" April 29,2002 http://www.sans.org/rr/firewall/cisco_pix.php

[2] Setting up PIX Syslog, http://www.cisco.com/warp/public/110/pixsyslog.html

[3] Cisco PIX Firewall System Log Messages, Version 6.2 http://www.cisco.com/univercd/cc/td/doc/product/iaabu/pix/pix_62/syslog/index.html

[4] Monitoring Cisco PIX Firewall with Syslog through a VPN Tunnel http://www.cisco.com/warp/public/110/pix_vpn_4094.html

[5] Wilson, Curt: "Cisco PIX attack patterns research", November 3, 2000 http://www.sans.org/y2k/110300.htm

[6] Bird, Tina and Ranum, Marcus: Log Analysis Resources http://www.loganalysis.org

[7] Cisco PIX Messages Listed by Severity Level http://www.cisco.com/univercd/cc/td/doc/product/iaabu/pix/pix_62/syslog/pixemapa.htm

[8] Cisco PIX Command Reference, Version 6.2 http://www.cisco.com/univercd/cc/td/doc/product/iaabu/pix/pix_62/cmdref/

Assignment Part #2: Network Detects

Practical Detect #1 -- Suspicious IGMP Packets

Posted to intrusions-at-incidents.org on January 3, 2003

0. Detect

```
[**] [1:527:3] BAD TRAFFIC same SRC/DST [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/13-08:22:18.726507 207.166.38.167 - 207.166.38.167
IGMP TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:28
[Xref = cve CVE-1999-0016]
[**] [1:527:3] BAD TRAFFIC same SRC/DST [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/13-08:22:18.726507 207.166.38.172 - 207.166.38.172
IGMP TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:28
[Xref = cve CVE-1999-0016]
[**] [1:527:3] BAD TRAFFIC same SRC/DST [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/13-08:22:18.726507 207.166.38.177 - 207.166.38.177
IGMP TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:28
[Xref = cve CVE-1999-0016]
```

...and some more (33 alerts in total).

```
giac@creosote:~ tcpdump -neX -r 2002.10.13 igmp
08:22:18.726507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 60: 207.166.38.167  207.166.38.167:
igmp query v2 [gaddr 240.0.3.146]
0x0000   4500 001c 0000 0000 2f02 37d8 cfa6 26a7        E......./.7...&.
0x0010   cfa6 26a7 1164 fb08 f000 0392 0000 0000        ..&..d..........
0x0020   0000 0000 0000 0000 0000 0000 0000             ..............
08:22:18.726507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 60: 207.166.38.172  207.166.38.172:
igmp query v2 [gaddr 240.0.3.151]
0x0000   4500 001c 0000 0000 2f02 37ce cfa6 26ac        E......./.7...&.
0x0010   cfa6 26ac 1164 fb03 f000 0397 0000 0000        ..&..d..........
0x0020   0000 0000 0000 0000 0000 0000 0000             ..............
08:22:18.726507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 60: 207.166.38.177  207.166.38.177:
igmp query v2 [gaddr 240.0.3.156]
0x0000   4500 001c 0000 0000 2f02 37c4 cfa6 26b1        E......./.7...&.
0x0010   cfa6 26b1 1164 fafe f000 039c 0000 0000        ..&..d..........
0x0020   0000 0000 0000 0000 0000 0000 0000             ..............
```

1. Source of Trace

http://www.incidents.org/logs/Raw, File 2002.10.13

2. Detect was generated by

Snort 1.9.0 on Linux, with the standard ruleset, snortrules-stable, obtained from www.snort.org on 02.JAN.2003. The command entered was: snort -d -c snort.conf -l /home/giac/2002.10.13.alerts -r /home/giac/2002.10.13

The actual rule triggering the alert is shown below.

```
alert ip any any - any any (msg:"BAD TRAFFIC same SRC/DST"; sameip; reference:cve,CVE-
1999-0016; reference:url,www.cert.org/advisories/CA-1997-28.html; classtype:bad-
unknown; sid:527; rev:3;)
```

This rule matches on all IP packets where the source and destination IP address happen to be identical (keyword "sameip" in the rule). The rule does not check any other conditions, like for an established connection or for a particular direction of traffic flow, which makes sense in view of the fact that a packet with identical source and destination addresses will not appear on the wire as part of a "normal" connection.

The cross references listed in the snort rule suggest that this rule has been added to catch the ages-old "Land Attack", a denial of service attack involving identical source and destination addresses. See section 4) below for more on this.

3. Probability that the source address was spoofed

IGMPv2 and v3, the Multicast Group Management Protocol defined in RFC 2236 [9] and RFC 3376 [10], specifies three types of IGMP messages:

- Membership Query
- Membership Report
- Leave Group Message

The packets at hand are IGMP membership queries, as apparent from both the protocol field in the IP header (0x02 = IGMP) and the IGMP Message Type field in the first byte of the payload (0x11 = Query).

IGMP Queries are issued by routers, in order to determine multicast membership status on an attached network. The protocol incorporates an "election process" which ensures that only one router per physical network (the "designated querier") will issue such requests.

The RFC distinguishes between two types of IGMP queries. A "General Query" is sent by the Desingated Querier to elicit responses from all active multicast participants on the segment, whereas a "Group Specific Query" is used to find out if a particular multicast group still has active members. General Queries are sent to the "all-systems" multicast address of 224.0.0.1, whereas specific queries are sent to the particular multicast address of the group. Since these queries are local to the attached network, the RFC specifies that they be sent with a time-to-live (TTL) of one (1).

Taking a closer look at the packets at hand, we discover that

- the packets are "Generic Group Queries", containing the group being queried for as part of the payload
- the TTL is 47
- the destination address is not a multicast address
- the MAC layer destination addresses is not a multicast address, either

From this, I conclude that the packets cannot be valid IGMP Group Queries and are either "crafted" or have been improperly obfuscated as part of the log cleanup process. This does not yet answer the question whether the _source_ has been spoofed, though.

To answer this with some certainty, we have to extend the scope of the investigation and stop looking only at the IGMP packets. From inspection of the remainder of the traffic in the log, it appears as if all traffic passing the IDS sensor only involves two distinct MAC layer addresses:

```
00:00:0C:04:B2:33
00:03:e3:d9:26:c0
```

This can be confirmed by running TCPdump against the log file and by filtering for everything but these addresses:

```
giac@creosote:~ /usr/sbin/tcpdump -neX -r 2002.10.13 not ether host 00:03:e3:D9:26:C0
| wc -l
0
```

From this circumstance, and from the fact that the vendor specific part of both MAC addresses indicates a Cisco device, I conclude that the IDS sensor is likely placed between two routers, as follows:

INTERNET -*- ROUTER1 ---T--- ROUTER2 -*- INSIDE

with the "T" being the Tap or connection point for the IDS. There might be additional devices like firewalls connected into the path at the locations indicated by a "*", but from the information contained in the logs, there's no telling for sure.

Traffic originating on the outside should always appear at the IDS sensor with the source MAC of ROUTER1 and the destination MAC of ROUTER2. From evaluating the remainder of the traffic in the log, I conclude that the inside network is using addresses from the 207.166.0.0/16 range. Filtering for the MAC addresses involved in communication TO the inside, we get:

```
giac@creosote:~ /usr/sbin/tcpdump -ne -r 2002.10.13 dst net 207.166.0.0/16 | awk
'{print $2 " - " $3}' | sort -u
0:3:e3:d9:26:c0 - 0:0:c:4:b2:33
```

Thus, we have established that ALL traffic with a destination on the inside network seems to originate from MAC 0:3:e3:d9:26:c0 (ROUTER1) and is headed for 0:0:c:4:b2:33 (ROUTER2).

For the opposite direction, the reverse does not seem to hold true:

```
giac@creosote:~ /usr/sbin/tcpdump -ne -r 2002.10.13 src net 207.166.0.0/16 | awk
'{print $2 " - " $3}' | sort -u
0:0:c:4:b2:33 - 0:3:e3:d9:26:c0
0:3:e3:d9:26:c0 - 0:0:c:4:b2:33
```

Traffic originating on the inside network seems to "flow" in both directions on the MAC level. But wait - this BPF filter also includes the IGMP packets which claim to come from the inside. What if we exclude them:

```
giac@creosote:~ /usr/sbin/tcpdump -ne -r 2002.10.13 '(not igmp) and (src net
207.166.0.0/16)' | awk '{print $2 " - " $3}' | sort -u
0:0:c:4:b2:33 - 0:3:e3:d9:26:c0
```

Bingo! It seems as if only the IGMP packets are violating the direction of the flow on the MAC layer. Consequently, these packets only APPEAR to come from the inside, but in fact are originating from the outside.

Bottom Line: The source address of these packets has been spoofed.

4.& 5. Description of the Attack / Attack Mechanism

I have no idea :-). Most known IGMP DoS attacks seem to involve the spoofing of IGMP membership reports [11,12], and not, as in this case, membership queries. This makes sense, since a membership report message directly influences multicast routing on the subnet. A query - normally issued by a router - is mainly used to ascertain that no multicast users have left the stream without signing off.

The device getting the spoofed IGMP query will likely take it at face value, though. In clarification of the somewhat obfuscated language used in the older standard, the new IGMPv3 RFC [10] clearly states that an IGMP-enabled system must process all IGMP queries addressed to any of the multicast OR UNICAST addresses valid for the interface on which the packet arrives.

In other words, according to the IGMP specification, our spoofed queries are perfectly OK. Consequently, the IGMP protocol stack will respond to the query as specified in the RFC, but only if the client is indeed member of the multicast group specified in the query (remember, it's a group-specific query and not a general query). Thus, in most cases, the device receiving the spoofed packet will simply do: nothing.

For the unlikely event that a response is necessary, IGMP responses are also being sent to multicast addresses. This means that the originally spoofed source address will NOT become the destination of the response packet, but will simply be discarded. Thus, the chances of the client getting confused by having to send a packet to himself are slim. The Land Attack [CVE-1999-0016], for which the original Snort alert of this detect was generated, is based on just this "confusion", and, more importantly, has been known (and patched) for years. Therefore I conclude that the intention of the packets is not a DoS against the receiving host.

Since multicast enabled routers attached to the same subnet elect a "designated querier" based on the IGMP queries that they themselves receive, the purpose of the spoofed packets could be to throw this election process into a spin, with the result being that no designated querier is left on the subnet and no _real_ membership queries are issued anymore. The timeouts built into the election process (see RFC) appear to be pretty robust - in order to do any real damage, I surmise that a constant stream of forged queries would be necessary, and not just meager 33 packets.
In order to be more conclusive in the analysis, it would be necessary to capture the entire traffic to/from the targeted systems and to evaluate if the queries elicit any response.

6. Correlations

Quite a few of these particular IGMP packets can be found in the various log files under http://www.incidents.org/logs/Raw, but so far I have failed to catch any of these packets "in the wild". Most likely due to the somewhat limited searchability :-) of the incidents mailing list, I could not locate any previous posts or practical assignments on the subject of using IGMP queries as an attack mechanism. Which doesn't mean there aren't any - please let me know.

7. Evidence of Active Targeting

A handful of these packets can be found in most of the 2002.10.x log files available on incidents.org. I therefore conclude that these packets are either the result of active targeting, a persistent misconfiguration of an IGMP device, or an artefact resulting from the log obfuscation process.

8. Severity

Criticality: 1
The IGMP packets are the only packets to/from these particular systems that have been picked up by the IDS. It is therefore impossible to deduct the importance of the targeted systems (or if they even exist, for that matter).

Lethality: 1
Unless these packets are the traces of a rare and not very well known attack method, I believe that the traffic will not cause any harm. If the packets are an attempt to use IGMP to stage a Land Attack, I believe that this will not work both due to the IGMP specification and due to the fact that most systems are immune against Land nowadays.

System Countermeasures: 3
No information available, which either means that the system countermeasures are good enough to withstand whatever these packets are trying to do -- or that the systems fell over and died right away, without giving any further evidence of their existence. Therefore: an average of three.

Network Countermeasures: 2
The entire logfile suggests that either the NIDS sensor has been placed close to the outer network perimeter, or that the network countermeasures are quite permissive. The absence of the otherwise pretty common SNMP, SMTP, Telnet and SSH alerts from the logs suggests that some sort of filtering must be in place, though.

Severity = 1+1 - 3+2 = -3 = don't lose too much sleep over this.

9. Defensive Contermeasures

An antispoofing filter on the outer network perimeter would surely be nice to have. And hardly any companies I know are actually pulling in multicast traffic from the Internet. Most likely, IGMP could be blocked on the perimeter as well.

10. Multiple Choice Question

Which of the following protocols do you expect to encounter in combination with Multicast traffic?

a) TCP
b) FTP
c) IGMP
d) ICMP

Answer: c)

11. Comments received on this analysis

Some of the comments received from readers of the intrusions-at-incidents.org mailing list are shown below. An excerpt of my responses is quoted inline in *italics.*

Carl Gibbons,cgibbons-at-du.edu, wrote

```
Carl> I saw these IGMP detects as well, and your analysis is very
Carl> thorough. It's my understanding that it's not just routers
Carl> (layer 3 devices) that communicate IGMP traffic, but Cisco
Carl> switches (layer 2 devices) also get involved in group
Carl> membership multicast protocols as well, so that multicast
Carl> traffic is only pushed to the switch ports that register for
Carl> it.
```

Myself> correct. but the packets at hand are fake IGMP membership
Myself> queries, as opposed to membership reports. forwarding
Myself> decisions on a switch are, as far as i can tell, influenced
Myself> by the latter rather than the former. membership queries are
Myself> issued by routers in order to locate active members of a
Myself> certain multicast group - thus i'd expect the switch to
Myself> ignore the query, but keep an eye peeled for the replies.

```
Carl> So your illustration of the sensor between two routers might
Carl> also look like this, with the sensor between a router and
Carl> a switch:
Carl>
Carl> INTERNET -*- ROUTER ---T--- SWITCH -*- INSIDE
Carl>
Carl> In other words, even though there are IGMP packets going both
Carl> directions, couldn't one source be an inside multicast host
Carl> connected to the inside switch, and not another router?"
```

Myself> possible. even the MAC addrs could be spoofed, of course.
Myself> but i don't consider this scenario to be very likely, as the
Myself> IGMP packets are the only ones in the whole log file which
Myself> violate the direction of the flow on layer 2. and switches
Myself> do not _actively_ participate in multicast traffic, therefore
Myself> seeing a switch either as a source or a sink (on layer 2) of
Myself> IGMP traffic would be even more evidence that something fishy

Myself> is going on.

Ronny Rietveld, ronny-at-plcrietveld.demon.nl, wrote

```
Ronny> Cisco CCNP Switching Exam Certification Guide, p353: 'Layer 2
Ronny> switches can snoop IGMP queries and reports to learn the port
Ronny> mappings of multicast groupmembers.'
Ronny> According to this, I would say that (Cisco) switches listen to
Ronny> both queries and reports.
```

Robert Wagner, rwagner-at-eruces.com, wrote

```
Robert> Defense Recommendations:
Robert> "An antispoofing filter on the outer network perimeter"
Robert> Is this the technical term?  Or do you mean ingress and
Robert> egress filters.
```

Myself> i'm not a native english speaker. the main reason why i
Myself> picked the term "antispoofing filter" is that there seems
Myself> to be some sort of ingress filtering in place, but it
Myself> apparently does NOT contain the particular ingress filters
Myself> needed to protect against spoofing. in other words and in my
Myself> understanding, anti-spoofing filters are (should be) a subset
Myself> of ingress and egress filtering.

```
Robert> You give the lethality of this being low (1). Can you make
Robert> this assumption without knowing the router type and version?
Robert> Is it possible that the router hasn't been patched in years
Robert> and is susceptible to the attack?
```

Myself> hmmm. the packets are addressed at end systems and not at
Myself> routers. my lethality rating is refering to the impact i
Myself> expect this "attack" to have on the end systems... but if
Myself> you are hinting at the fact that an unpatched router might
Myself> blow up from forwarding this illegal packet or from copying
Myself> it into the router's register of other IGMP queriers on the
Myself> subnet -- yes, you're right. i'm still sticking to my rating
Myself> of (1), though, as I haven't found an indication on the web
Myself> that such a vulnerability indeed exists or existed.

Practical Detect #2 -- NetBios Password Attempt

Posted to intrusions-at-incidents.org on January 11, 2003

0. Detect

```
giac@creosote:~ /usr/sbin/tcpdump -nXr 2002.10.16 port 139
12:36:54.126507 218.151.67.21.1210  170.129.50.16.139: P 1448655:1448714(59)
ack 4178294196 win 8756NBT Packet (DF)
0x0000   4500 0063 7244 4000 6a06 a412 da97 4315        E..crD@.j.....C.
0x0010   aa81 3210 04ba 008b 0016 1acf f90b b5b4        ..2.............
0x0020   5018 2234 9814 0000 0000 0037 ff53 4d42        P."4.......7.SMB
0x0030   7500 0000 0000 0000 0000 0000 0000 0000        u...............
0x0040   0000 0000 0000 0000 0000 0000 04ff 0000        ................
0x0050   0000 0001 000c 0021 5c5c 4232 425c 4300        .......!\\B2B\C.
0x0060   413a 00                                        A:.
12:55:40.566507 217.227.208.29.2618  170.129.50.16.139: P 4144552:4144611(59)
ack 163480733 win 8572NBT Packet (DF)
0x0000   4500 0063 a976 4000 7506 d58b d9e3 d01d        E..c.v@.u.......
0x0010   aa81 3210 0a3a 008b 003f 3da8 09be 849d        ..2..:...?=.....
0x0020   5018 217c 045b 0000 0000 0037 ff53 4d42        P.!|.[.....7.SMB
0x0030   7500 0000 0000 0000 0000 0000 0000 0000        u...............
0x0040   0000 0000 0000 0000 0000 0000 04ff 0000        ................
0x0050   0000 0001 000c 0021 5c5c 4232 425c 4300        .......!\\B2B\C.
0x0060   413a 00                                        A:.
14:39:07.196507 213.123.77.13.1029  170.129.50.16.139: P 22775469:22775528(59)
ack 1706499141 win 8188NBT Packet
0x0000   4500 0063 08ac 0000 0f06 a3cf d57b 4d0d        E..c.........{M.
0x0010   aa81 3210 0405 008b 015b 86ad 65b7 1c45        ..2......[..e..E
0x0020   5018 1ffc 55c6 0000 0000 0037 ff53 4d42        P...U......7.SMB
0x0030   7500 0000 0000 0000 0000 0000 0000 0000        u...............
0x0040   0000 0000 0000 0000 0000 0000 04ff 0000        ................
0x0050   0000 0001 000c 0021 5c5c 4232 425c 4300        .......!\\B2B\C.
0x0060   413a 00                                        A:.
```

1. Source of Trace

http://www.incidents.org/logs/Raw, File 2002.10.16 As previously reported by other students, the file label does not tie in with the timestamps of the records in the file. It appears as if the data in this file has been recorded on November (not October) 16th, 2002.

2. Detect was generated by

The README file available at http://www.incidents.org/logs/Raw states that the trace files located in the same directory were generated by Snort running in binary logging mode, with an unpublished ruleset. This suggests that ALL packets logged must have violated the unpublished Snort ruleset, as they simply would not be in the file otherwise.

Running snort-1.9.0 gainst the 2002.10.16 trace file, I noticed that "my" instance of Snort was far from issuing an alert for EVERY packet contained in the original log. This incited my curiosity to find out what "my" Snort was missing and why.

First, I modified the original snort.conf to disable the stream4 reassembly engine and to turn on all the signatures which are disabled by default. Then, using this new config file, I ran the following commands:

```
snort -d -b -A fast -c snort.conf-everything -l log1016 -r 2002.10.16
```

By having Snort read in the original log file ("-r") and writing out again those packets that actually triggered an alert when compared to the "snort.conf-everything" ruleset, I ended up with two binary files - the original and the result from my processing.

```
-rw-r--r--    1 giac     users      451320 2003-01-05 16:02 2002.10.16
-rw-------    1 giac     users      299178 2003-01-05 16:06 snort.log.1041779196
```

Running these two files through "tcpdump -nr" to turn them back into ASCII logs and subsequently comparing the two ASCII files with "diff" resulted in an ASCII tcpdump output containing all those packets that _did_ match the unknown ruleset used to create the original trace, but which _failed_ to match on my instance of snort. An excerpt of this output is shown below.

```
64.154.80.50.80: P 3266503306:3266504038(732) ack 500951635 win 17520 (DF)
64.154.80.50.80: P 1529417926:1529418657(731) ack 2765552405 win 33580 [tos 0x10]
66.35.229.104.80: P 47715891:47716912(1021) ack 332915483 win 16229 [tos 0x10]
170.129.50.3.80: P 231347558:231348098(540) ack 3376890412 win 32246 [tos 0x10]
170.129.50.16.139: P 1448655:1448714(59) ack 4178294196 win 8756 NBT Packet (DF)
170.129.50.16.139: P 4144552:4144611(59) ack 163480733 win 8572 NBT Packet (DF)
208.33.48.101.80: P 4267449296:4267450076(780) ack 3172806183 win 17520 (DF)
208.33.48.101.80: P 3200324776:3200325555(779) ack 1094643895 win 33580 [tos 0x10]
208.33.48.101.80: P 4267767193:4267767991(798) ack 1589037064 win 17520 (DF)
208.33.48.101.80: P 1616238010:1616238807(797) ack 2678730929 win 33580 [tos 0x10]
```

From closer examination, it seems as if there's only two categories of packets which _fail_ to trigger an alert with the regular snort rules. One set involves web traffic, whereas the other contains SMB/Netbios traffic.

I decided to take a closer look at the SMB traffic. The web traffic is left as an exercise to the inclined reader :-). The SMB detect shown above was created by filtering the original log for SMB packets only:

```
giac@creosote:~ /usr/sbin/tcpdump -nXr 2002.10.16 port 139
```

3. Probability that the source address was spoofed

All three packets are part of an established TCP connection and come with all the fixings of arbitrary sequence and acknowledgement numbers, varying TTLs, push and ack flags, and source ports in the expected non-privileged range. Of course, all of this could be falsifed - but then I would expect to see additional signs of packet crafting like a constant source port or somesuch shared between the three detects. This is not the case. From this, and from the fact that whoever is trying to issue a SMB connect would most likely also prefer to see the response of his/her actions, I conclude that the source addresses are NOT spoofed.

The detect involves three source addresses, namely 218.151.67.21, 217.227.208.29 and 213.123.77.13. An excerpt from the whois information is shown below. None of the addresses are listed in the DShield database.

```
IP Address         : 218.151.67.0-218.151.67.127
Connect ISP Name   : KORNET
Network Name       : KORNET-LLINE-IKSAN-NETELPC

inetnum:      217.224.0.0 - 217.237.161.47
netname:      DTAG-DIAL15
descr:        Deutsche Telekom AG

inetnum:      213.123.74.0 - 213.123.77.255
netname:      BT-IMSNET
descr:        BT Public Internet Service
```

4. Description of Attack

The packets in the detect are evidence of an attempt to mount shared drive C: on a Windows system whose NetBios name is "B2B" (\\B2B\C). The packets in question are the stimulus, there's no telling from the log whether the attempt is successful.

The standard Snort rules are only checking for illicit access to the Windows default shares (C$/D$/ADMIN$/IPC$) and hence do not catch this attempt. By modifying one of the existing NetBios Snort rules (file netbios.rules) from

```
alert tcp $EXTERNAL_NET any - $HOME_NET 139 (msg:"NETBIOS SMB C$ access"; flow:
to_server,established; content: "|5c|C$|00 41 3a 00|";reference:arachnids,339;
classtype:attempted-recon; sid:533; rev:5;)
```

into a rule catching all access to the C drive share

```
alert tcp $EXTERNAL_NET any - $HOME_NET 139 (msg:"NETBIOS SMB C access";
flow:to_server,established; content: "|5c|C|00 41 3a 00|";reference:arachnids,339;
classtype:attempted-recon; sid:100533; rev:5;)
```

it is possible to generate an alert out of the three packets looking as follows:

```
[Classification: Attempted Information Leak] [Priority: 2]
11/16-12:36:54.126507 218.151.67.21:1210 - 170.129.50.16:139
TCP TTL:106 TOS:0x0 ID:29252 IpLen:20 DgmLen:99 DF
***AP*** Seq: 0x161ACF  Ack: 0xF90BB5B4  Win: 0x2234  TcpLen: 20
[Xref = arachnids 339]
[**] [1:100533:5] NETBIOS SMB C access [**]
[Classification: Attempted Information Leak] [Priority: 2]
11/16-12:55:40.566507 217.227.208.29:2618 - 170.129.50.16:139
TCP TTL:117 TOS:0x0 ID:43382 IpLen:20 DgmLen:99 DF
***AP*** Seq: 0x3F3DA8  Ack: 0x9BE849D  Win: 0x217C  TcpLen: 20
[Xref = arachnids 339]
[**] [1:100533:5] NETBIOS SMB C access [**]
[Classification: Attempted Information Leak] [Priority: 2]
11/16-14:39:07.196507 213.123.77.13:1029 - 170.129.50.16:139
TCP TTL:15 TOS:0x0 ID:2220 IpLen:20 DgmLen:99
***AP*** Seq: 0x15B86AD  Ack: 0x65B71C45  Win: 0x1FFC  TcpLen: 20
[Xref = arachnids 339]
```

5. Attack Mechanism

The fact that three distinct source addresses are attempting to access the very same NetBios share (\\B2B\C) suggests one of two things. Either, all three attackers are using the same tool, into which the NetBios and Share name have been hard-coded. Or, and this is more likely, the system being targeted responds to SMB protocol negotiation requests. These requests, also sent over port 139, trigger a response containing (among other information) the netbios name. Since none of the standard Snort rules seem to match on SMB Negotiate Protocol packets (type 0x72), it is to be expected that these packets, if present, would not show up in the log.

The second particularity that all three access attempts have in common is that they do not constitute an "anonymous" access attempt (Password 0x00), but rather initiate the connection with a one-byte password of "!" (0x21), as shown below in the excerpt of Ethereal output. The user ID field is set to zero, though, which is an indication that the user has not previously established an authenticated session with the server. Hence, the password provided is likely a "share level" (as opposed to: individual) password, a protection mechanism which is mainly used with Windows 9x and ME.

```
        Reserved: 000000000000000000000000
        Tree ID: 0
        Process ID: 0
        User ID: 0
        Multiplex ID: 0
    Tree Connect AndX Request (0x75)
        Word Count (WCT): 4
        AndXCommand: No further commands
        Reserved: 00
        AndXOffset: 0
        Flags: 0x0000
            .... .... .... ...0 = Disconnect TID: Do NOT disconnect
        Password Length: 1
        Byte Count (BCC): 12
        Password: 21
        Path: \\B2B\C
        Service: A:
```

Non-null passwords with a length of one byte could suggest that this is an attempt to exploit the years-old SMB Share Level Password vulnerability (MS00-072 / CVE-2000-0979 / BID 1780). In brief, systems with this vulnerability erroneously only verify the first byte of the password when provided with a password of length one. Connection requests with a password of "!" would therefore be successful on a share whose password had been set to "!PaSSw0RD". Brute-forcing a one-byte password is obvioulsy both easy and feasible. See [13] for a description of the vulnerability.

As this vulnerability has been known for quite some time, I was intrigued to find evidence of three different attackers trying it out within minutes against the same system. Too much of a coincidence. Consequently, I went looking for attack tools or worms which could have caused this pattern. And found a likely match in W32.Opaserv aka W95/Scrup, a worm that first appeared in early October 2002.

The following worm description was taken from F-Secure.com [14]. The page also contains a detailled description of the effects the worm and its variants (can) have on affected systems:

> In order to find victim computers Opasoft scans subnets for port 137 (NETBIOS Name Service). IP addresses of the following networks are scanned:
>
> - current subnet of the infected computer (aa.bb.cc)
> - the two nearest subnets of the currently infected computer (aa.bb.cc+1 , aa.bb.cc-1)
> - selects subnets randomly (excluding those where scanning is disabled)
>
> If while searching (scanning) Opasoft happens upon a responding IP address (of an actual computer), the worm then scans the two nearest subnets of that IP address. When "reply data" is received, Opasoft checks the special field that it contains. If it shows that the given computer has the service "File and Print Sharing" open, Opasoft begins its infection procedure on that computer as a remote host.
>
> During infection, Opasoft sends, via port 139 (NETBIOS Session Service) special SMB - packets that transmit the following commands:
>
> 1. sets a connection with the \\hostname\C resource, where "hostname" = the name of the victim computer which is defined when the victim computer answers Opasoft (by sending its "reply data") during the scan
> 2. if the resource is password-protected the worm runs through all possible "one symbol" passwords - conducting a "brute-force" attack
> 3. if connection is successful, Opasoft transmits its EXE file - during transmission the full name of the destination file containing the code (exe file) is revealed

From this description, W32.Opasoft sure looks like a possible cause for the packets at hand. The one mismatch between the description and our packets is that the packets do not contain any evidence that indeed a brute-force attempt of all one-symbol passwords is taking place. The exclamation mark "!" in the password field of all three logged packets could suggest that this is the first packet of the brute force attack (with "!" being the first printable character of the ASCII alphabet). The reason why the remaining probes do not appear in the log could be that a) the worm code is flawed b) the non-standard Snort rule only matches on the "!" packet c) the password indeed starts with "!" and the probe is successful on the first try

From the evidence and information at hand, I'm unable to tell which of these options happens to be the case.

6. Correlations

SANS Critical Vulnerability Newsletter, dated 7OCT02, with a warning on the OpaServ/OpaSoft worm. http://www.sans.org/newsletters/cva/cva1_11.php

Virus Statistics from Trend Micro clearly show a peak around the time when the detect was recorded.

Message to the DShield List dated 13Nov02, reporting a OpaServ infection http://www.dshield.org/pipermail/list/2002-November/001641.html

Discussion on Snort-Sigs Mailing List on how to detect OpaServ http://sourceforge.net/mailarchive/forum.php?thread_id=1350339&forum_id=7141 The patterns suggested in this thread can only be used to detect an existing infection and hence are not the patterns used at the site from where the detect was obtained.

7. Evidence of Active Targeting

None, if the packets were indeed caused by a worm (which I believe). Some of the OpaServ variants have a tendency to go after "neighbouring" class C networks by incrementing or decrementing the corresponding portion of the IP address of the infected host. This still cannot be called active targeting, though, it's more like "living in an unsafe neighbourhood" :-).

8. Severity

Criticality = 3
Hard to tell. The only time the targeted system shows up in the trace is with the three packets that make up this detect. The NetBios name "B2B" could suggest, though, that the device is involved in some sort of Business-to-Business transaction, which would make it a critical resource indeed.

Lethality = 4
The various OpaServ variants come complete with backdoor code and a timebomb trojan. The worm will also spread to any other vulnerable system on the inside and outside networks. Hence, the resulting damage could be serious.

System Countermeasuers = 2
It seems as if File and Printer Sharing is active on the targeted system and has not been locked down to prevent access. From the evidence in the logs, it is hard to tell whether the system is patched against the one-byte password vulnerability or not.

Network Countermeasures = 1
Poor. SMB protocol negotiation and tree connect packages are simply not something one should let in through the firewall coming from arbitrary hosts. (not let in, period, for that matter).

Severity = (3+4)-(2+1) = 4 = better get moving

9. Defensive Recommendation

Apply ingress and egress filters on the outer network perimeter in order to block NetBios/SMB traffic over ports 135, 137 and 138 UDP as well as 135, 139 and 445 TCP. If the site is indeed running some sort of eBusiness/B2B service, it might be a good idea to switch from a "allow unless denied" type of ruleset to the more restrictive "deny unless allowed" behavior by adding a "deny any any" rule at the end of the ruleset.

10. Multiple Choice Test Question

The port and protocol used to connect to a NetBios share on a Windows computer is

a) 139/tcp
b) 139/udp
c) 137/tcp
d) 137/udp

Answer: a), the NetBios Session Service running on Port 139/tcp

Practical Detect #3 -- SSH Version Map Attempt

Posted to intrusions-at-incidents.org on January 19, 2003

0. Detect

```
01/14-05:10:22.619014  [**] [1:1638:3] EXPERIMENTAL SCAN SSH Version map attempt
[**] [Classification: Detection of a Network Scan] [Priority: 3] {TCP}
203.237.115.179:3827 - 192.168.16.8:22
01/14-22:02:24.557060  [**] [1:1638:3] EXPERIMENTAL SCAN SSH Version map attempt
[**] [Classification: Detection of a Network Scan] [Priority: 3] {TCP}
213.35.172.50:1852 - 192.168.16.8:22

giac@creosote:~> tcpdump -tttt -vv -nXr 20030114_3 host 213.35.172.50
01/14/2003 22:02:23.247520 213.35.172.50.1728 > 192.168.16.8.22: S [tcp sum ok]
2660688647:2660688647(0) win 32120 <mss 1400,sackOK,timestamp 292282120,nop,wscale 0>
(DF) (ttl 48, id 27903, len 60)
0x0000   4500 003c 6cff 4000 3006 8bb6 d523 ac32        E..<l.@.0....#.2
0x0010   c0a8 1008 06c0 0016 9e96 e307 0000 0000        ................
0x0020   a002 7d78 f1dd 0000 0204 0578 0402 080a        ..}x.......x....
0x0030   01bd fcb4 0000 0000 0103 0300                  ............
01/14/2003 22:02:23.249250 192.168.16.8.22 > 213.35.172.50.1728: S [tcp sum ok]
1115682077:1115682077(0) ack 2660688648 win 5792 <mss 1460,sackOK,timestamp 16847319
29228212,nop,wscale 0> (DF) (ttl 64, id 2880, len 60)
0x0000   4500 003c 0b40 4000 4006 dd75 c0a8 1008        E..<.@@.@..u....
0x0010   d523 ac32 0016 06c0 427f f51d 9e96 e308        .#.2....B.......
0x0020   a012 16a0 0df4 0000 0204 05b4 0402 080a        ................
0x0030   0101 11d7 01bd fcb4 0103 0300                  ............
01/14/2003 22:02:23.466094 213.35.172.50.1728 > 192.168.16.8.22: . [tcp sum ok]
1:1(0) ack 1 win 32120 <nop,nop,timestamp 29228234 16847319> (DF) (ttl 48, id 27958,
len 52)
0x0000   4500 0034 6d36 4000 3006 8b87 d523 ac32        E..4m6@.0....#.2
0x0010   c0a8 1008 06c0 0016 9e96 e308 427f f51e        ............B...
0x0020   8010 7d78 d5ca 0000 0101 080a 01bd fcca        ..}x............
0x0030   0101 11d7                                      ....
01/14/2003 22:02:23.470712 192.168.16.8.22 > 213.35.172.50.1728: P [tcp sum ok]
1:24(23) ack 1 win 5792 <nop,nop,timestamp 16847341 29228234> (DF) (ttl 64, id 2881,
len 75)
0x0000   4500 004b 0b41 4000 4006 dd65 c0a8 1008        E..K.A@.@..e....
0x0010   d523 ac32 0016 06c0 427f f51e 9e96 e308        .#.2....B.......
0x0020   8018 16a0 cf9c 0000 0101 080a 0101 11ed        ................
0x0030   01bd fcca 5353 482d 312e 3939 2d4f 7065        ....SSH-1.99-Ope
0x0040   6e53 5348 5f33 2e34 7031 0a                    nSSH_3.4p1.
01/14/2003 22:02:23.688612 213.35.172.50.1728 > 192.168.16.8.22: . [tcp sum ok]
1:1(0) ack 24 win 32120 <nop,nop,timestamp 29228257 16847341> (DF) (ttl 48, id 27962,
len 52)
0x0000   4500 0034 6d3a 4000 3006 8b83 d523 ac32        E..4m:@.0....#.2
0x0010   c0a8 1008 06c0 0016 9e96 e308 427f f535        ............B..5
0x0020   8010 7d78 d586 0000 0101 080a 01bd fce1        ..}x............
0x0030   0101 11ed                                      ....
01/14/2003 22:02:24.100172 213.35.172.50.1852 > 192.168.16.8.22: S [tcp sum ok]
2671568410:2671568410(0) win 32120 <mss 1400,sackOK,timestamp 29228298 0,nop,wscale 0>
(DF) (ttl 48, id 27964, len 60)
0x0000   4500 003c 6d3c 4000 3006 8b79 d523 ac32        E..<m<@.0..y.#.2
0x0010   c0a8 1008 073c 0016 9f3c e61a 0000 0000        .....<...<......
0x0020   a002 7d78 ed52 0000 0204 0578 0402 080a        ..}x.R.....x....
0x0030   01bd fd0a 0000 0000 0103 0300                  ............
01/14/2003 22:02:24.100349 192.168.16.8.22 > 213.35.172.50.1852: S [tcp sum ok]
1122574028:1122574028(0) ack 2671568411 win 5792 <mss 1460,sackOK,timestamp 16847404
29228298,nop,wscale 0> (DF) (ttl 64, id 2882, len 60)
0x0000   4500 003c 0b42 4000 4006 dd73 c0a8 1008        E..<.B@.@..s....
0x0010   d523 ac32 0016 073c 42e9 1ecc 9f3c e61b        .#.2...<B....<..
0x0020   a012 16a0 defb 0000 0204 05b4 0402 080a        ................
0x0030   0101 122c 01bd fd0a 0103 0300                  ...,........
01/14/2003 22:02:24.330562 213.35.172.50.1852 > 192.168.16.8.22: . [tcp sum ok]
1:1(0) ack 1 win 32120 <nop,nop,timestamp 29228321 16847404> (DF) (ttl 48, id 27965,
len 52)
0x0000   4500 0034 6d3d 4000 3006 8b80 d523 ac32        E..4m=@.0....#.2
0x0010   c0a8 1008 073c 0016 9f3c e61b 42e9 1ecd        .....<...<..B...
0x0020   8010 7d78 a6d1 0000 0101 080a 01bd fd21        ..}x...........!
0x0030   0101 122c                                      ...,
01/14/2003 22:02:24.335234 192.168.16.8.22 > 213.35.172.50.1852: P [tcp sum ok]
1:24(23) ack 1 win 5792 <nop,nop,timestamp 16847428 29228321> (DF) (ttl 64, id 2883,
len 75)
0x0000   4500 004b 0b43 4000 4006 dd63 c0a8 1008        E..K.C@.@..c....
0x0010   d523 ac32 0016 073c 42e9 1ecd 9f3c e61b        .#.2...<B....<..
0x0020   8018 16a0 a0a1 0000 0101 080a 0101 1244        ...............D
0x0030   01bd fd21 5353 482d 312e 3939 2d4f 7065        ...!SSH-1.99-Ope
0x0040   6e53 5348 5f33 2e34 7031 0a                    nSSH_3.4p1.
01/14/2003 22:02:24.551114 213.35.172.50.1852 > 192.168.16.8.22: . [tcp sum ok]
1:1(0) ack 24 win 32120 <nop,nop,timestamp 29228343 16847428> (DF) (ttl 48, id 27966,
len 52)
0x0000   4500 0034 6d3e 4000 3006 8b7f d523 ac32        E..4m>@.0....#.2
0x0010   c0a8 1008 073c 0016 9f3c e61b 42e9 1ee4        .....<...<..B...
0x0020   8010 7d78 a68c 0000 0101 080a 01bd fd37        ..}x...........7
0x0030   0101 1244                                      ...D
01/14/2003 22:02:24.557060 213.35.172.50.1852 > 192.168.16.8.22: P [tcp sum ok]
1:29(28) ack 24 win 32120 <nop,nop,timestamp 29228343 16847428> (DF) (ttl 48, id
27967, len 80)
0x0000   4500 0050 6d3f 4000 3006 8b62 d523 ac32        E..Pm?@.0..b.#.2
0x0010   c0a8 1008 073c 0016 9f3c e61b 42e9 1ee4        .....<...<..B...
0x0020   8018 7d78 3fee 0000 0101 080a 01bd fd37        ..}x?..........7
0x0030   0101 1244 5353 482d 312e 302d 5353 485f        ...DSSH-1.0-SSH_
0x0040   5665 7273 696f 6e5f 4d61 7070 6572 0a00        Version_Mapper..
01/14/2003 22:02:24.557183 192.168.16.8.22 > 213.35.172.50.1852: . [tcp sum ok]
24:24(0) ack 29 win 5792 <nop,nop,timestamp 16847450 29228343> (DF) (ttl 64, id 2884,
len 52)
0x0000   4500 0034 0b44 4000 4006 dd79 c0a8 1008        E..4.D@.@..y....
0x0010   d523 ac32 0016 073c 42e9 1ee4 9f3c e637        .#.2...<B....<.7
0x0020   8010 16a0 0d33 0000 0101 080a 0101 125a        .....3.........Z
0x0030   01bd fd37                                      ...7
01/14/2003 22:02:24.558884 192.168.16.8.22 > 213.35.172.50.1852: F [tcp sum ok]
24:24(0) ack 29 win 5792 <nop,nop,timestamp 16847450 29228343> (DF) (ttl 64, id 2885,
len 52)
0x0000   4500 0034 0b45 4000 4006 dd78 c0a8 1008        E..4.E@.@..x....
0x0010   d523 ac32 0016 073c 42e9 1ee4 9f3c e637        .#.2...<B....<.7
0x0020   8011 16a0 0d32 0000 0101 080a 0101 125a        .....2.........Z
0x0030   01bd fd37                                      ...7
01/14/2003 22:02:24.559016 213.35.172.50.1852 > 192.168.16.8.22: F [tcp sum ok]
29:29(0) ack 24 win 32120 <nop,nop,timestamp 29228343 16847428> (DF) (ttl 48, id
27968, len 52)
0x0000   4500 0034 6d40 4000 3006 8b7d d523 ac32        E..4m@@.0..}.#.2
0x0010   c0a8 1008 073c 0016 9f3c e637 42e9 1ee4        .....<...<.7B...
0x0020   8011 7d78 a66f 0000 0101 080a 01bd fd37        ..}x.o.........7
0x0030   0101 1244                                      ...D
[some packets deleted to save space]
```

1. Source of Trace

Snort 1.9.0 running as NIDS on a perimeter network of a small company. The sensor is installed on the inside of a Cisco PIX firewall, which translates the outside routable address range to the 192.168.16.0/27 addresses used on the DMZ. Hence, no obfuscation of the logs was needed to "protect" the innocent. The target is a Linux 2.4.19 system running an Apache web server and OpenSSH 3.4p1. I am not affiliated with this company, but was asked by a friend working there to verify their installation. Intrigued by the fact that they had several hundred megabytes of "real" tcpdump archives (instead of the partial Snort dumps I am used to deal with), I asked them for permission to use their logs for my practical.

2. Detect was generated by

Snort 1.9.0 running as a NIDS behind a puny but powerful Cisco PIX 501 Firewall. The rules used by this Snort instance were last updated Oct 22, 2002. The experimental signature which triggered this detect has by now been included into the standard Snort rulebase. The sensor is also running an instance of tcpdump to collect the full traffic to/from the perimeter network for certain ports and protocols. Due to the space requirements, these logs are being rotated automatically and overwritten as needed.

3. Probability that source address was spoofed

The attacker completes a valid 3-way handshake, exchanges information with the target, and then gracefully closes the connection. Hence, the attacker's address cannot be spoofed.

4. Description of Attack

The attacker conducted a scan of the entire subnet, but since the system shown in the trace is the only one reachable by means of SSH from the outside, all other attempts were blocked by the firewall. The attack consists of two parts, one being used to locate systems offering the SSH service, and the second being an attempt to map the exact version of SSH used. The information thus gathered can then be used to stage an attack against those (quite numerous) versions of SSH and OpenSSH which are vulnerable. See http://www.openssh.org/security.html and several CERT advisories like [15] for details on vulnerable versions. Depending on the version of SSH used, vulnerabilities range from denial of service to remotely exploitable buffer overflows.

5. Attack mechanism

The SSH Version scanner "ScanSSH" first appeared in September 2000. About one year later, in October 2001, a paper [16] describing the functionality was published by Niels Provos and Peter Honeyman at the University of Michigan. Source code for ScanSSH is available from Monkey.org. From the paper, it appears as if the tool was initially conceived as software for vulnerability research on the university campus. Since the code is well designed and written, though, it is not surprising that ScanSSH now also finds use in a research community with less beningn intentions.

ScanSSH consists of two processes, called "producer" and "consumer". The producer process is used to scan entire address ranges for systems whose SSH port is reachable. The consumer process then picks up this list and connects again to the same systems in order to pull down the SSH version banner. While this approach to fingerprinting could be straight out of a software engineering textbook, it still strikes me as somewhat silly to connect _twice_ in order to get the version banner -- but at least this seemingly odd behaviour makes this particular type of scan easy to detect. The second tell-tale sign for the use of SSHScan is, of course, the calling card left by the software with its identification string SSH-1.0-SSH_Version_Mapper. This is the string on which the Snort rule triggered.

```
alert tcp $EXTERNAL_NET any - $HOME_NET 22 (msg:"EXPERIMENTAL SCAN SSH Version
map attempt"; flow:to_server,established; content:"Version_Mapper"; nocase;
classtype:network-scan; sid:1638; rev:3;)
```

The alleged SSH version number used in this identification string (1.0) is not a valid SSH versoin number, and will cause all existing SSH servers to disconnect (but only after revealing the server version to the scanner). Newer versions of SSHD will take a client version number of 1.0 as a sign that they have just been probed by an SSH Version Mapper and will record this fact to syslog.

According to Toby Miller's "OS Fingerprinting" paper [17], it looks from the typical characteristics of the SYN packets like the attacker is using Linux with a kernel from the 2.2 series (both the options sled and the window of 32120 are indications to that effect). The low initial MSS used might be a sign that the attacker is using a DSL modem and PPPoE.

```
01/14/2003 22:02:24.100172 213.35.172.50.1852  192.168.16.8.22: S [tcp sum ok]
2671568410:2671568410(0) win 32120 <mss 1400,sackOK,timestamp 29228298
0,nop,wscale 0> (DF) (ttl 48, id 27964, len 60)
```

6. Correlations

The first attacker is listed with DShield as originating from Korea, with 34 incriminating records for attempts on ports 443 and 21 (but not 22).
http://www.dshield.org/ipinfo.php?ip=203.237.115.179

The second attacker is also listed with DShield and originates from Estonia.
http://www.dshield.org/ipinfo.php?ip=213.35.172.50
At MyNetWatchman, this source is also listed as a recent scanner for SSH versions: http://www.mynetwatchman.com/LID.asp?IID=18531144

7. Evidence of Active Targeting

None - looks like a simple scan to gather version information. Evidence of active targeting would be if an attacker came back at a later date to attempt an exploit, but no such evidence has been found in the logs for the past few days.

8. Severity

Criticality = 3
The target hosts the public website of a small company and is as such of some importance. A quick inquiry revealed that SSH access to the box had been left open "so that the web designer can upload the new pages".

Lethality = 1
The recorded attempts are only probing activity and no actual attack.

System Countermeasures = 4
OpenSSH version 3.4p1 is reasonably up to date and does not contain any well-known vulnerabilities. If some sort of TCPwrappers are in use, they are apparently not configured to strike on connections to port 22.

Network Countermeasures = 3
From the remaining log entries, it becomes apparent that the targeted system

is only reachable from the outside over port 22 and 80, all other connections are apparently being blocked by the firewall. Also, there is no evidence in the Snort log that ssh access (tcp/22) is allowed to any other system in the DMZ. The fact that both Snort and a traffic logger are running is an indication that somebody has given the question of network defense some thought. This favorable impression is somewhat offset by the outdated Snort rules on the sensor. Seems as if whoever installed this NIDS has by now lost interest in the upkeep.

Severity = (3+1) - (4+3) = -3 = none too scary

9. Defensive Recommendation

I question the wisdom of having SSH open "from everywhere" only because the external web designer employed by the company apparently cannot afford a fixed IP address for his business and gets teleported through the IP space whenever his ISP feels like it. As a minimum, I would try to filter inbound SSH to only include the web designer's ISP's address range(s). If that's not feasible for some reason, I would consider adding some fudge factor to the problem by having the PIX translate the port as well, and thus "relocate" the SSH instance away from tcp/22 into some obscure corner of the tcp high ports. While this does not help against a dedicated adversary, it makes it fare less likely that tools like "ScanSSH" stumble over a vulnerable server by accident. And,yes, upgrading that wizened Snort ruleset could not hurt, either.

10. Multiple Choice Test Question

The string SSH-1.0-SSH_Version_Mapper is used by SSHScan to
a) elicit a version banner response from the mapped SSH server
b) switch to SSH protocol 1.0 to map the old version number
c) leave a calling card and cause the server to disconnect, because SSH protocol number 1.0 is not specified
d) start the "CRC32 Compensation Attack" on the SSH daemon

Answer: c).
SSH protocol 1.0 is not specified, which will cause older versions of SSHD to disconnect. Newer versions of SSHD will disconnect and record the fact that a potential SSHScan mapping has been attempted to the syslog.

Assignment Part #3: Intrusion Log Analysis : 15 - 19 Feb, 2003

Executive Summary

A passive network security audit was conducted for MYNET.edu, a renowned research university in Baltimore, MD.[1]. The audit was performed in a completely non-invasive manner, basing on five consecutive days worth of intrusion detection log files provided to the analyst by MYNET.edu staff. As requested by the customer, *no* active scanning or perimeter vulnerability assessments have been performed against MYNET.edu systems.

The data used for the analysis covers the time from Saturday, February 15, 2003, up to and including Wednesday, February 19, 2003. These dates were chosen to ensure that the raw data contains both weekend and weekday activity patterns.

In the course of the analysis, evidence was uncovered which suggests that three systems on the MYNET.edu campus have been compromised by the Adore/Red Worm and are now being remotely controled by parties connecting from outside of the MYNET.edu network.

We also found evidence pointing towards that 19 (nineteen) MYNET.edu systems have been compromised and are offering their services and storage space publicly to selected user groups over Internet Relay Chat (IRC).

Further, the data analyzed contains proof that numerous students and/or employees of MYNET.edu are actively engaged in the exchange of audio and video data. The wealth of evidence leads us to believe that file sharing over so-called "peer-to-peer" networks is a favorite pastime at MYNET.edu, and may well amount to a sizeable portion of the University's internet traffic. Since the audio and video files shared over these networks often include copy-righted material, MYNET.edu could be faced with lawsuits or liability issues.

The high number of alerts issued by the MYNET.edu intrusion detection system suggests that the system is not very well maintained and not kept up to date. Intrusion detection systems should be carefully tuned to only issue alerts at a rate which can actually be processed in near-realtime by an intrusion analyst. If the system was mainly built to collect statistical data or evidence for later analysis in case of a break-in, some adjustments to the configuration should be made to make the system better fit these purposes.

We recommend that MYNET.edu management should take steps to define, in writing, the purpose of and requirements on the MYNET.edu intrusion detection system. Further, we suggest that MYNET.edu management should provide for adequate staffing in order to ensure the implementation and upkeep of the aforementioned requriements, as well as timely response to detected incidents.

[1]It is obvious from the scans log data, which was not properly sanitized, that the university in question is the University of Maryland, Baltimore County (umbc.edu). For the reminder of this assignment, we stick to the "sanitized" terminology and call the insitution MYNET.edu.

Introduction

A passive network security audit was conducted for MYNET.edu, a renowned research university in Baltimore, MD. The audit was performed in a completely non-invasive manner, basing on five consecutive days worth of intrusion detection log files provided to the analyst by MYNET.edu staff. As requested by the customer, no active scanning or perimeter vulnerability assessments have been performed against MYNET.edu systems.

The data used for the analysis covers the time from Satuarday, February 15, 2003, up to and including Wednesday, February 19, 2003. These dates were chosen to ensure that the raw data contains both weekend and weekday activity patterns.

The intrusion detection system used by MYNET.edu maintains three different log files for each calendar day. The "alerts" file contains intrusion alert log entries in basic Snort format[2], the "scans" file contains the logs of Snort's portscan preprocesser, and finally the "OOS" file, which is used to log headers or packet content of traffic which violates certain specifications of the TCP/IP protocol. The table below shows the names of the actual raw data files provided by MYNET.edu for the analysis. All files are, at the time of writing, also available for download at http://isc.incidents.org/logs .

Date	*Alert File*	*Scans File*	*OOS File*
15/02/2003	alert.030215.gz	scans.030215.gz	OOS_Report_2003_02_16_32309
16/02/2003	alert.030216.gz	scans.030216.gz	OOS_Report_2003_02_17_6137
17/02/2003	alert.030217.gz	scans.030217.gz	OOS_Report_2003_02_18_27913
18/02/2003	alert.030218.gz	scans.030218.gz	OOS_Report_2003_02_19_479
19/02/2003	alert.030219.gz	scans.030219.gz	OOS_Report_2003_02_20_28598

Table 1 Log files used in the analysis

The graph shown on the next page serves to provide a quick overview on the activity that was logged during the analysis period. Note that the scale of the vertical axis is logarithmic in order to accomodate the huge difference between the average number of scans, alerts and out-of-spec packets per hour. While alert activity remains fairly constant at around 1000 events per hour, scanning activity was found to oscillate by serveral orders of magnitude between roughly 15'000 to well below 100. Out-of-spec (OOS) data is being logged at a comparably leisurly pace of about sixty packets per hour.

From the sheer number of logged events alone it is apparent that the Intrusion Detection System (IDS) at MYNET.edu is likely not used in the course of the daily work of an Intrusion Analyst. The system, in its current form, appears to hover somewhere between being an alerting instrument and a tool to collect statistical information, but has apparently not been fine-tuned in some time to perfectly fit either purpose. Please refer to the chapter titled

[2]Snort appears to be running in "Fast" Alert mode, with a custom (non-standard) rule set. In the course of this paper, it is assumed that the reader is at least marginally familiar with Snort, the Open Source Network Intrusion Detection System. For more information, refer to the Snort documentation [18]

"Defensive Recommendations" later in this document for some suggestions on how to address this.

While the graph suggests that scanning activity is by far the highest on a Sunday (Feb 16) and has its daily low around 4 am in the morning, this could be mere coincidence. The five days worth of log files do not provide sufficient material for a statistical analysis of the event distribution over time.

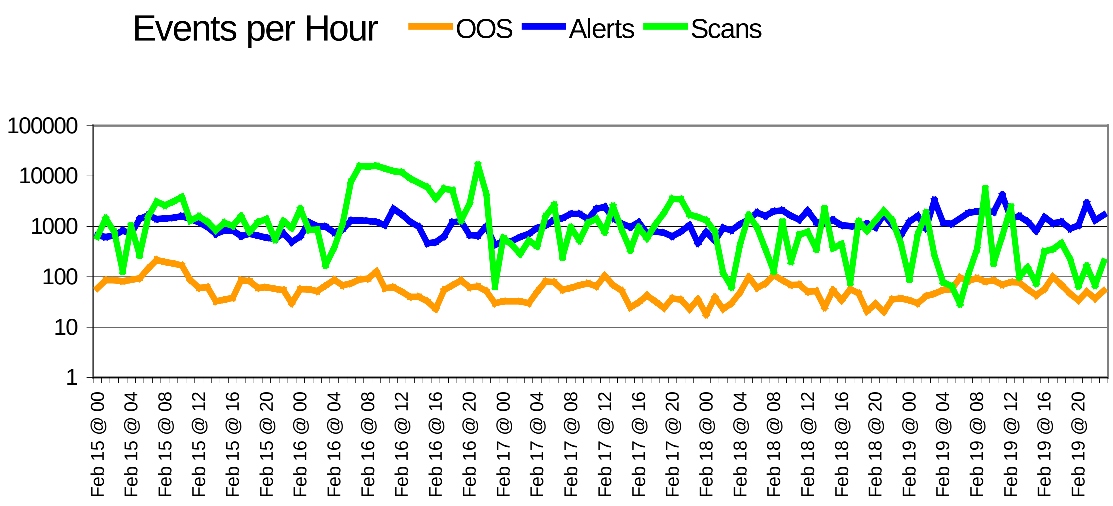


Illustration 1 Time line of analyzed events

Triage and Analysis of Log Data

In order to get an overview over the "average" activity on the MYNET.edu networks, the log files of each class ("alert", "scans" and "OOS") were concatenated into one huge file each, and then passed through several small scripts and filter expressions to extract "data of interest". Cutting and dicing the huge amount of data in five different ways, as shown below, allows us to quicky get an impression on where the problems might lay and to prioritize the more detailed analysis which is to follow. We refer to this initial screening process as "triage".

The five log processing steps used in the initial triage round are

1. Identification of systems on the inside network which are acting as servers
2. Identifying systems and services which appear to cause the most alerts
3. Identifying the most dangerous alerts
4. Locating the sources of heavy scanning activity
5. Identifying systems issuing out-of-spec packet data

In a second round, the more interesting results of these initial triage steps were then analyzed further. Some of the actual queries and filter expressions used can be found at the end of this document in a separate chapte titled "Data Reduction and Analysis Techniques".

Step One - Identifying "servers" on the Inside

This step aims to identify systems on the inside network which seem to act as servers. By filtering the alert data for sources on the inside network (*outbound* traffic) and then tallying the frequency by which a certain source port occurrs for any given source address, it is possible to locate those systems which seem to have a "certain affinity" to a particular source port. Since outgoing client connections are supposed to use arbitrary (random) source ports, a system showing a clear preference for a certain source port is likely a server and the logged packet a reply, not a request. As a second approach, all alerts of *inbound* traffic to the MYNET.edu network were evaluated with regards to how frequent a certain destination port was sought after per inside system.

Among the results of this analysis step, we identified some of the official MYNET.edu server systems, like public websites and systems used to forward and store MYNET.edu email. These servers are of interest since they are most visible and often serve as a first point of contact or prime target for an attacker[3]. Almost as a side effect, the filter also turned up a number of "unofficial" or "clandestine" servers and potential backdoors on the inside network.

The resulting list of suspected servers was then cut down to only show servers either using privileged source ports (< 1024) or servers with more than thirty (30) events for a particular source port. These arbitrary restrictions were introduced, at the risk of missing an "interesting" system or two, to keep the resulting data within manageable limits. Listed below are some systems which showed "server-like" behaviour when subjected to this initial screening and were selected for further analysis.

Suspected Backdoors

Source Address	# Entries	Alert(s) Logged	Severity
MY.NET.244.246 MY.NET.246.178 MY.NET.235.10	81 96 40	[**] High port 65535 tcp - Possible Red Worm [**]	**high**

Several external systems were logged connecting to port 65535/tcp on these three MYNET.edu machines, and all three systems were also captured when responding to requests. It can therefore be concluded with certainty that all three systems are indeed running some sort of server on port 65535. The last two systems also show up in the data collected by the "Out of Spec" packet sensor, but the traffic logged there only relates to Gnutella file sharing activity into which these two systems seem to be involved as well. Since no actual packet data of connections to the suspected backdoor was captured or retained by the MYNET.edu IDS sensors, we can only assume (but not prove) that these three systems have indeed been infected by RedWorm/Adore and have a backdoor shell active on port 65535. The sheer number of external systems (more than eighty!) which were found to be communicating with said port could also suggest that the three MYNET.edu

[3] Normally, information on these systems would be available to the analyst as part of an engagement, but the data can - at least partially - also be deducted from the log files.

systems are running some sort of file sharing service on this unusual high port. If the port is indeed bound to a backdoor, then the backdoor is being very widely used. More information on the Redworm/Adore Backdoor can be found at [19].

Suspected Peer-to-Peer Filesharing Servers

Using or providing a peer-to-peer (P2P) or file sharing service is not a dangerous activity per se. But from the evidence collected by the MYNET.edu IDS sensors, we conclude that these services are being *massively* used by MYNET.edu staff and students and as such might well impair the bandwith available to productive University data traffic. Also, file sharing networks are often employed to exchange or trade copyrighted audio and video material. Some partial packets logged by the OOS IDS filter suggest that his is also the case at MYNET.edu. Consequently, MYNET.edu should ensure that staff and students have been informed, in writing, that sharing of copyrighted data over the University networks is prohibited. Without such an "acceptable use" guideline, MYNET.edu could open itself to liability or copyright infringment lawsuits.

It seems as if no dedicated IDS rules to monitor the use of Peer-to-Peer tools are in place at MYNET.edu. Thus, most of the activity outlined below was picked up almost "by accident", by other rules which are generic enough to sometimes also (usually mistakenly) trigger on P2P traffic.

KaZaA Servers

Source Address	# Entries	Alert(s) Logged	Severity
MY.NET.212.22	559	[**] Port 55850 tcp - Possible myserver activity [**] [**] Wachtlist 000220 IL-ISDNNET-990517 [**]	medium

The actually logged alerts are not of relevance with regard to this detect. Much more interesting than the actual alerts is the fact that both types of alerts were caused by four external systems (12.234.50.23, 24.245.42.53, 65.27.250.93, 212.179.72.34) which all happened to be talking to TCP port 2046 on the MYNET.edu system.This suggests that MY.NET.212.22 is either deliberately running some sort of clandestine server (gaming, file sharing, etc), or that the system has been compromised and a backdoor is present on that port.

Without actual packet data of a connection to/from this system, it would have been impossible to decide on either file sharing or backdoor. Luckily, the "out of spec" processor captured one entire packet headed for MY.NET.212.22 which allowed us to tilt the balance towards "KaZaA" file sharing server [20]. The packet excerpt is shown below.

```
02/16-10:13:53.306503 212.38.46.78:2015 -> MY.NET.212.22:1844
TCP TTL:113 TOS:0x0 ID:25817 IpLen:20 DgmLen:386 DF
*2UAPRSF Seq: 0x96E0AA47  Ack: 0x4213010D  Win: 0x8FE9  TcpLen: 12
UrgPtr: 0x0
47 45 54 20 2F 2E 68 61 73 68 3D 64 34 64 61 32  GET /.hash=d4da2
```

```
39 38 33 65 30 38 63 32 32 38 63 30 66 31 39 36   983e08c228c0f196
32 32 30 30 66 61 38 65 61 38 63 64 32 35 66 34   2200fa8ea8cd25f4
66 64 64 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F   fdd HTTP/1.1..Ho
73 74 3A 20 31 33 30 2E 38 35 2E 32 31 32 2E 32   st: MY.NET.212.2
32 3A 32 30 34 36 0D 0A 55 73 65 72 41 67 65 6E   2:2046..UserAgen
74 3A 20 4B 61 7A 61 61 43 6C 69 65 6E 74 20 4D   t: KazaaClient M
61 79 20 32 38 20 32 30 30 32 20 31 34 3A 35 31   ay 28 2002 14:51
3A 32 31 0D 0A 58 2D 4B 61 7A 61 61 2D 55 73 65   :21..X-Kazaa-Use
72 6E 61 6D 65 3A 20 4A 61 73 6F 6E 4B 61 79 49   rname: JasonKayI
74 61 6C 79 0D 0A 58 2D 4B 61 7A 61 61 2D 4E 65   taly..X-Kazaa-Ne
74 77 6F 72 6B 3A 20 66 69 6C 65 73 68 61 72 65   twork: fileshare
0D 0A 58 2D 4B 61 7A 61 61 2D 49 50 3A 20 31 30   ..X-Kazaa-IP: 10
2E 30 2E 30 2E 31 34 39 3A 31 32 31 34 0D 0A 58   .0.0.149:1214..X
2D 4B 61 7A 61 61 2D 53 75 70 65 72 6E 6F 64 65   -Kazaa-Supernode
49 50 3A 20 31 34 39 2E 31 35 39 2E 31 32 31 2E   IP: 149.159.121.
38 35 3A 31 35 36 37 0D 0A 52 61 6E 67 65 3A 20   85:1567..Range:
62 79 74 65 73 3D 34 30 35 36 38 37 35 39 30 2D   bytes=405687590-
35 37 31 34 31 37 30 39 36 0D 0A 43 6F 6E 6E 65   571417096..Conne
63 74 69 6F 6E 3A 20 63 6C 6F 73 65 0D 0A 58 2D   ction: close..X-
4B 61 7A 61 61 2D 58 66 65 72 49 64 3A 20 31 32   Kazaa-XferId: 12
38 34 33 38 31 38 0D 0A 0D 0A                     843818....
```

While this packet is also evidence that somebody from Italy (212.38.46.78) is playing nasty with the MYNET.edu system by bugging it with an out-of-spec TCP packet with all TCP flags turned on, the payload of the packet thus collected is clear proof that MY.NET.212.22 is indeed running a KaZaA server on the non-standard port 2046.

Source Address	# Entries	Alert(s) Logged	Severity
MY.NET.206.242 MY.NET.234.14	1284 245	[**] Watchlist 000220 IL-ISDNNET-990517 [**] [**] Possible trojan server activity [**]	medium

Again, the actual alerts logged are of little relevance with regard to this detect. Both systems are on the receiving end of serveral connection attempts to TCP port 1214, which is the "standard" port of the KaZaA File Sharing tool. The second system, MY.NET.234.14, has also been logged with several responses originating from port 1214 and is therefore definitely running a server process on that port.

EDonkey2000 Servers

Source Address	# Entries	Alert(s) Logged	Severity
MY.NET.222.174 MY.NET.237.66 MY.NET.220.106	1243 39 42	[**] Watchlist 000220 IL-ISDNNET-990517 [**] [**] Queso fingerprint [**]	medium

Likely due to an excessive number of false positives, the "Queso" fingerprint alert has long since been removed from the standard Snort rulebase. But as expected, the alleged Queso packets (OS fingerprinting with unusual TCP flags) are also featured prominently in the logs of the out-of-spec packet sensor. From these packets, we can conclude that the three systems above are indeed involved in EDonkey2000 [21] file sharing transactions and have an EDonkey server service running on TCP port 4662.

WinMX Servers

Source Address	# Entries	Alert(s) Logged	Severity
MY.NET.70.176	55	[**] High port 65535 udp - Possible Red Worm [**]	medium
MY.NET.224.210	71		
MY.NET.227.118	1		
MY.NET.84.178	81		
MY.NET.83.146	72		
MY.NET.235.178	54		
MY.NET.221.130	51		

WinMX [22] seems to be actively scanning the net for other WinMX instances by probing target servers for UDP port 6257. Since some of these connections apparently happen to use UDP port 65535 as source, these connections were caught in spades by the RedWorm filter (which is completely unrelated to WinMX).

Gnutella/BearShare Servers

Source Address	# Entries	Alert(s) Logged	Severity
MY.NET.202.50		[**] Queso fingerprint [**]	medium

Again, the Queso pattern is picking up traffic which is in fact pretty benign in nature. MY.NET.202.50 is on the receiving end of various attempts to connect to the Gnutella/BearShare [23] port tcp/6346, but has not been logged to actually respond to these requests. Nevertheless, evidence retrieved from the out-of-spec packet log confirms that this system is repeatedly acting as a Gnutella client.

TFTP Clients

Source Address	# Entries	Alert(s) Logged	Severity
MY.NET.237.238	908	[**] TFTP - Internal TCP conn to ext tftp server [**]	low
MY.NET.223.114	870		

The listed two systems are originating numerous outbound TFTP connections to 64.12.29.76, 64.12.30.136, 64.12.28.4, 64.12.27.224 and 64.12.25.148 and are exchanging data with those systems, all of which reside in the AOL address space (64.12.x). No evidence regarding the type of data being exchanged could be gained from the available logs, but we nevertheless recommend that MYNET.edu staff examine the nature of these connections.

Step Two - Systems and Services causing the most alerts

As a second step, we simply tallied how often a particular system or port showed up in the alert logs, both as a source and as a destination. While this step contributes only little information to triage process, it serves well to locate particularly "noisy" systems and services. Systems and services which are topping the statistic are either indeed broken or compromised, or the particular alert rules of the intrusion detection system falsely match on benign traffic. Both conditions should be rectified.

Top Alert Sources

Source Address	# Entries	Alert(s) Logged	Severity
MY.NET.211.6	13192	[**] Incomplete Packet Fragments Discarded [**]	low

This system is exchanging information with two external systems, 198.247.231.42 and 216.111.123.20, and seems to suffer from some fragmentation problems. Since these packets are the only ones logged as originating from MY.NET.211.6, the exact role of that system cannot be determined. The external parties are using an address from the VERIO ISP netblock (198.247.*) and an address assigned to "Creative Internet Technologies", a firm located in Powell, OH. No obvious evidence of malicious intent, but the problems persist over the entire analysis period and suggest some sort of misconfiguration on the side of MY.NET.211.6 or the external parites.b

Source Address	# Entries	Alert(s) Logged	Severity
169.232.84.146	4726	[**] SUNRPC highport access [**]	low

The listed system is hammering away at port 32771 on the internal host MY.NET.252.126. Since the inside system has not been logged as responding to these requests, the alerts could be disregarded. But a quick verification of all connections to MY.NET.252.126 revealed a certain interest into port 32771 by various external sources. Since no apparent packet filtering is in place on the standard SunRPC port (111), access attempts to this secondary port are of little additional consequence. If the targeted system *is* located behind some sort of filtering device, it should be verified whether both ports 111 and 32771 have been blocked, and/or whether the targeted system has been patched against this pretty old (1997) Sun Solaris vulnerability. See [24] for details.

Source Address	# Entries	Alert(s) Logged	Severity
212.179.123.163	2508	[**] Watchlist 000220 IL-ISDNNET-990517 [**]	medium

The listed system is connecting repeatedly to MY.NET.235.62 on port 1321. Neither evidence of responses from the target system nor other activity of the target are available from the log files. Port 1321 is commonly being used by Informix Database servers. The alert rule "Watchlist 000220" seems to trigger on all traffic originating from the Israelian network range 212.179.*. It is not clear why this specific alert rule has been added to the IDS, but the date (990517) suggests that it might have been used years ago to closely monitor hostile activity. This particular actvity is not of obvious malicious intent, but

MYNET.edu should nevertheless verify which service MY.NET.235.62 is running on port 1321 and whether this service has been properly protected.

Source Address	# Entries	Alert(s) Logged	Severity
12.35.158.199	1779	[**] SMB Name Wildcard [**] [**] NETBIOS NT NULL session [**]	**high**

The listed system is probing wide ranges of the MY.NET.* network, accessing 1779 distinct systems within roughly 80 (eighty!) hours and looking for open Netbios ports. The slow and careful manner of the probing successfully evades the scan detection preprocessor. Those internal systems which were apparently responsive on port 137 are then subsequently contacted on the Netbios session service port (139), with a connection attempt using an empty ("NULL") password. The extent of this activity and its stealthy nature have led us to classify it with a severity of "high".

```
whois -h whois.arin.net MCTAN656-158-192

OrgName:    Mckenzie Tankline
OrgID:      MCKENZ-2
Address:    122 appleyard drive
City:       tallahasse
StateProv:  FL
PostalCode: 32304
Country:    US

NetRange:   12.35.158.192 - 12.35.158.207
CIDR:       12.35.158.192/28
NetName:    MCTAN656-158-192
NetHandle:  NET-12-35-158-192-1
Parent:     NET-12-0-0-0-1
NetType:    Reassigned
Comment:
RegDate:    2000-12-21
Updated:    2000-12-21

TechHandle: KR52-ARIN
TechName:   Raquel, Key
TechPhone:  +1-850-576-1221
TechEmail:  mckenzietanklines@worldnet.att.net
```

According to the Whois information listed above, the source addess belongs to a range assigned to "McKenzie Tankline", an trucking outfit located in Tallahasse, Florida. Since we consider it to be unlikely that McKenzie staff is involved in probing attempts against MYNET.edu, we conclude that the system in question at McKenzie must have been compromised and is being abused by parties unknown. The address is not listed in the DShield database, and is only listed once at mynetwatchman.com, also as a scanner for Netbios services.

Source Addr	# Entries	Reason	Severity
MY.NET.222.174	24	[**] spp_http_decode: IIS Unicode attack [**] [**] TFTP - Internal TCP conn. to tftp server [**] [**] CGI Null Byte attack detected [**]	**high**

This system, which has already been identified as an EDonkey2000 File Sharing server in the previous analysis step, is initiating suspicous connections to a handful of external systems. The activity is not sustained enough to assume that the system is infected with a self-propagating worm like Nimda or CodeRed. From the timing pattern of the activity (excerpt shown below) we assume that user on the console is manually employing some tool to probe external webservers for the presence of Unicode and CGI vulnerabilities.

```
02/17-16:50:41.720819  [**] spp_http_decode: CGI Null Byte attack
detected [**] MY.NET.222.174:4033 -> 62.194.109.123:80
02/17-16:52:50.581959  [**] spp_http_decode: IIS Unicode attack
detected [**] MY.NET.222.174:4033 -> 62.194.109.123:80
02/17-21:49:50.341683  [**] spp_http_decode: CGI Null Byte attack
detected [**] MY.NET.222.174:2431 -> 68.65.144.85:80
02/17-22:00:49.981592  [**] spp_http_decode: IIS Unicode attack
detected [**] MY.NET.222.174:2431 -> 68.65.144.85:80
```

The static source port of this activity does not really tie in with the time passing between the two queries. The fact that the targeted web servers seem to be located in cable modem address space in the Netherlands and in Philadelphia, PA makes a dedicated break-in attempt appear pretty unlikely, though.

Top Alert Destinations

Destination Addr	# Entries	Reason	Severity
MY.NET.100.165	7577	[**] CS WEBSERVER - external web traffic [**]	Noise

A separate IDS rule for this server seems to trigger whenever the system is connected to from an outside address. There is no way to tell which (if any) of these connections are malicious in nature. If there indeed is a valid need to protect this server against external access, MYNET.edu should consider installing a firewall for that purpose.

Destination Addr	# Entries	Reason	Severity
216.209.164.171	1790	[**] TCP SRC and DST outside network [**]	**high**

System port 135 is receiving packets from 1790 different sources in the 171.165.* address range, all with varying source ports. Neither address range is on the inside network. A reverse DNS lookup of the destination address reveals that 216.209.164.171 is called "Newmarket-ppp277234.sympatico.ca" and apparently belongs to a dialup connection into Canadian internet provider. The source address space 171.165.* belongs to the Bank of America (Whois information listed on next page).

For the sake of clarity, a number of the offending log entries is shown below

```
02/19-21:52:07.909910  [**] TCP SRC and DST outside [**]
171.165.180.125:1473 -> 216.209.164.171:135
02/19-21:52:07.957504  [**] TCP SRC and DST outside [**]
```

```
171.165.180.134:1751 -> 216.209.164.171:135
02/19-21:52:08.037607  [**] TCP SRC and DST outside [**]
171.165.180.147:1357 -> 216.209.164.171:135
```

Assuming that routing is properly configured on the MYNET.edu network perimeter, traffic destined for 216.209.164.171 cannot end up on the campus network due to "natural" causes. This leaves the assumption that the traffic has its source on the inside network and, using spoofed source addresses, is headed outbound. The targeted system will (if port 135 is open) respond with a "syn ack" packet back to the spoofed Bank of America system, but the sustained packet rate is not high enough to cause any trouble on the side of BoA. Consequently, we conclude that the dialup system 216.209.164.171 is the true target of this activity, possibly an attempt to trigger the so-called "Spike" denial of service [25] vulnerability present in Windows2000 systems.

OrgName: Bank of America OrgID: BANKOF-2 Address: 2000 Clayton Road Address: M/S CA4-704-04-21 City: Concord StateProv: CA PostalCode: 94520 Country: US NetRange: 171.128.0.0 - 171.206.255.255 CIDR: 171.128.0.0/10, 171.192.0.0/13, 171.200.0.0/14, 171.204.0.0/15, 171.206.0.0/16 NetName: BAC-171-128-0-0-1 NetHandle: NET-171-128-0-0-1 Parent: NET-171-0-0-0-0 NetType: Direct Assignment NameServer: NS1.BANKOFAMERICA.COM NameServer: NS2.BANKOFAMERICA.COM NameServer: NS3.BANKOFAMERICA.COM NameServer: NS4.BANKOFAMERICA.COM Comment: RegDate: 1995-02-01 Updated: 2002-11-20 OrgTechHandle: ZB29-ARIN OrgTechName: hostmaster OrgTechPhone: +1-925-675-3744 OrgTechEmail: hostmaster@bankofamerica.com	CustName: HSE (Bell Nexxia) Address: 160 Elgin Street City: Ottawa StateProv: Ontario PostalCode: K2P 2C4 Country: CA RegDate: 2000-02-04 Updated: 2000-02-04 NetRange: 216.209.152.0 - 216.209.167.255 CIDR: 216.209.152.0/21, 216.209.160.0/21 NetName: HSE002-CA NetHandle: NET-216-209-152-0-1 Parent: NET-216-208-0-0-1 NetType: Reassigned Comment: RegDate: 2000-02-04 Updated: 2000-02-04

Table 2 - Whois information for the two involved parties

Destination Addr	# Entries	Reason	Severity
192.168.0.253	1493	[**] TFTP - External UDP conn. to tftp server [**]	mis-config

A misconfigured device, likely connected to the MY.NET.111.x subnet, is apparently issuing a steady stream of TFTP connection requests. Four TFTP servers, MY.NET.111.219, .230, .231and .232 respond tirelessly to these queries. Since the RFC 1918 address range of 192.168.x is indeed not part of the University campus network, these connections get erroneously logged as external attempts to contact TFTP servers on the inside.

Source and Destination Port Alert Statistics

The tables below are showing a tally of the ten most frequent source and destination ports which were found to trigger alerts.

# Entries	Source Port
10402	137
8879	1025
8280	1026
7532	1027
6082	1028
4730	2465
4539	1029
2600	69
2531	65535
2503	80

# Entries	Destination Port
74837	137
17669	80
5796	32771
4276	65535
2893	1214
2378	1321
1792	135
1613	4662
948	69
834	2708

Table 3 - Top ten source and destination port numbers causing alerts

While the source port statistics are showing the expected result of a network with mainly Windows clients using ephemeral source ports just above 1024, the list also reveals some interesting entries like the high number of occurences of Port 2465 as a source. This port did not show up as a suspected server service in the first triage step above, which suggests that the source of this traffic is outside of the MYNET.edu network. The tally of destination ports is mostly as expected as well. Since most IDS filter rules still focus on requests rather than replies, seeing many "well-known" server ports in this list is nothing out of the ordinary. This tentatively explains the "normal" server service ports like 137, 80, 135 and 69, but does not justify the high number of occurences of 1214,1321,2708,4662,32771 and 65535. The latter two can be explained with the fact that the IDS seems to contain specific rules triggering on port 32771 and 65535, to catch attempts at SunRPC high ports and to uncover RedWorm backdoors. From the other ports, we have also encountered 1214 and 4662 before, as tell-tale signs of KaZaA and EDonkey2000 peer-to-peer file sharing networks. Port 1321 we have already come across as suspected Informix DB server running on MY.NET.235.62. This leaves 2465 and 2708 for further investigation in detail

Port 2465

From the alert log excerpt shown below, it appears as if source port 2465 has been used for the massive barrage of SunRPC requests directed at MY.NET.252.126. This activity originating from 169.232.84.146 has already been listed earlier, when we inspected the top alert sources. Next to these 4700-something requests, the 8 (eight) other occurences of port 2465 as source port in the log files appear to be completely unrelated to this event. Hence, we conclude that port 2465 does not have any special significance and only appeared on our radar screen due to the statistical approach at selecting events.

```
02/19-03:30:00.790744  [**] SUNRPC highport access! [**]
169.232.84.146:2465 -> MY.NET.252.126:32771
02/19-03:30:01.102750  [**] SUNRPC highport access! [**]
169.232.84.146:2465 -> MY.NET.252.126:32771
02/19-03:30:01.103260  [**] SUNRPC highport access! [**]
169.232.84.146:2465 -> MY.NET.252.126:32771
```

Port 2708

Destination Addr	# Entries	Reason	Severity
MY.NET.245.106	834	[**] Watchlist 000220 IL-ISDNNET-990517 [**]	**high**

Port 2708 shows up in the alert logs only as a destination port on MY.NET.245.106. The alert triggering on these requests happens to be the one catching all traffic to/from the 212.179.x subnet in Israel, but this seems to be a "side effect" of the activity rather than the true story. Checking with the scans log file, we notice that MY.NET.245.106 is listed as a scanner and seems to target arbitrary systems, on arbitrary ports. An excerpt from the scans log is shown below.

```
Feb 18 18:35:35 MY.NET.245.106:2708 -> 24.167.44.70:1996 UDP
Feb 18 18:35:36 MY.NET.245.106:2708 -> 24.207.187.134:2909 UDP
Feb 18 18:35:37 MY.NET.245.106:2708 -> 24.160.77.202:3338 UDP
Feb 18 18:35:37 MY.NET.245.106:2708 -> 24.44.192.91:1505 UDP
Feb 18 18:35:37 MY.NET.245.106:2708 -> 66.26.245.137:2504 UDP
Feb 18 18:35:38 MY.NET.245.106:2708 -> 24.129.81.162:3431 UDP
Feb 18 18:35:38 MY.NET.245.106:2708 -> 24.209.10.128:1428 UDP
Feb 18 18:35:38 MY.NET.245.106:2708 -> 12.246.253.59:1363 UDP
Feb 18 18:35:38 MY.NET.245.106:2708 -> 24.168.111.130:1347 UDP
```

The direction of traffic shown in the scans log is most likely misleading - our interpretation of the activity is that the listed "targets" have indeed been contacting the MYNET.edu system on port 2708, and that said system has responded to these requests. Hence, it seems as if we have caught ourselves another "clandestine" server on the inside network, this one running on UDP port 2708. According to the IANA assigned port numbers list [26], 2708/udp seems to be related to "Banyan-Net / Vines", but - all nostalgic feelings aside - we don't really believe that the mentioned MYNET.edu system is indeed running this wizened network protocol. Since no other references to any tool or backdoor using this particular UDP port could be found on the net, we strongly suggest that MYNET.edu staff configure the IDS to capture entire packets of this activity in order to be able to analyze it further.

Step Three - Identify the most dangerous alerts

We use a tally of the actual alert messages for this third step of the initial data triage. It is important to note that the list shown below is *not* an excerpt of the most frequent alert messages, but actually shows *all* the alert messages which appear in the five days' worth of alert logs. The alert text of many of these messages suggests that the rule in question is not part of the standard

Snort IDS rulebase, but rather has been added or customized by MYNET.edu staff. And, alas, it is also apparent that some of this custom made rules are responsible for a lot of "clutter" in the alert files. Alert rules "firing" more than at most a couple of hundred times within five days do not provide any real value to the process of detecting intrusions. Either network defense measures (firewalls) should be improved to keep some of this traffic from ever reaching the inside network and IDS sensor, or the corresponding rules should be fine-tuned or disabled.

Some of the alert messages also suggest that MYNET.edu staff is using the IDS system almost like a firewall. As an example, if external web traffic to the "CS Webserver" (6548 hits) is considered to be evil, why is the traffic even allowed to reach the system in question? Also, if traffic from a certain subnet in Israel is of particular interest ("Watchlist 000220", 12615 hits), issuing an alert for every packet originating from said network will do little, if any, good.

As a consequence, we are unable to assign a severity level to many of these events without knowing the perceived or real threat which prompted MYNET.edu staff to manually add the corresponding rule to the IDS system. These entries are flagged with a severity of "as configured" in the table below, a label used to point out that the severity of the particular event is as high or as low as the person adding the rule perceived the threat.

Please note that the severity levels have been assigned to the alerts based on the *evaluation* of the actual log data and must *not* be read as a judgment towards the severity of an alert *in general.* An alert evaluated as being low in severity in the context of the analyzed MYNET.edu data might well be of "critical" severity elsewhere, depending on the presence of vulnerable systems and/or apparently successful attempts to exploit them.

# Entries	Alert Text and Explanation	Severity
74813	[**] SMB Name Wildcard [**] Since port 137 is apparently not filtered on the MYNET.edu perimeter or the IDS is attached on the outside of this filter, the SMB Name Wildcard IDS rule is catching and reporting the loads of scans for port 137 which are currently abound on the Internet	Noise
15072	[**] Incomplete Packet Fragments Discarded [**] Most of the alerts are generated by MY.NET.211.6 trying to talk to two external systems. Said system is likely either misconfigured or defective.	Config Error
12615	[**] Watchlist 000220 IL-ISDNNET-990517 [**] Lots of traffic on lots of different ports. IDS rule seems to catch and report all connections involving systems in the 212.179 subnet.	As configured
6548	[**] CS WEBSERVER - external web traffic [**] IDS rule seems to catch and report all external connections to this web/ftp server.	As configured

5889	[**] spp_http_decode: IIS Unicode attack detected [**] Almost half of the total figure of these alerts are caused by MYNET.edu systems connecting to systems in the 211.233.* network range of "www.daum.net". These Korean language web pages are apparently visited frequently by MYNET.edu users and some of the Korean language characters used on the page seem to trigger the Unicode alert rule.	Low
5834	[**] High port 65535 tcp - possible Red Worm - traffic [**] Most of the alerts are false positives triggering on the accidental use of port 65535 as ephemeral source port of a beningn conversation, but the alerts also caught three internal systems, MY.NET.246.178, MY.NET.244.246 and MY.NET.235.10, which seem indeed to run some sort of backdoor on port 65535.	**High**
5778	[**] SUNRPC highport access! [**] On first sight. all of these connections seem to be false positives triggered by the accidental use of 32771 as ephemeral source port in a connection.	Low
3478	[**] spp_http_decode: CGI Null Byte attack detected [**] While some of these alerts might be the real thing, most are caused by MYNET.edu users visiting web sites like 209.10.239.135, which turns out to be an old site of the movie database iFilm.com, where some content of the site seems to falsely trigger this alert.	Low
2737	[**] TCP SRC and DST outside network [**] Besides the hundreds of packets sent to 216.209.164.171:135 from systems in the 171.165.* range, this rule also catches a lot of misconfigured/unconfigured devices on the internal network using source addresses of 0.0.0.0 or addresses from RFC1918 ranges (192.168.x, etc) or the ZeroConfig AutoIP range (169.254.x)	Low
1946	[**] TFTP - Internal TCP connection to external tftp server [**]	Not analyzed
1493	[**] TFTP - External UDP connection to internal tftp server [**]	Not analyzed
1357	[**] Watchlist 000222 NET-NCFC [**] Alert rule seems to trigger on all packets involving 159.226.* as either source or destination. None of the 1357 alerts logged contains evidence of something apparently untoward happening.	As configured
971	[**] High port 65535 udp - possible Red Worm - traffic [**] According to all available documentaiton, the RedWorm backdoor is not using UDP, but TCP. Indeed, of all the packets caught by this rule, all seem to be pretty harmless in nature and appear to be the result of a lengthy WinMX file sharing session where one party happeded to use UDP/65535 as source port.	Noise / As configured
792	[**] Port 55850 tcp - Possible myserver activity ref. 010313 [**] The only information that Google uncovered on the "Myserver" DDOS agent suggests that myserver was/is indeed listening on port 55850, but only for UDP packets. This ties in with a quick glance through these alerts, most of which seem to be results of an alert rule triggering on the use of port 55850 as	Noise / As configured

	ephemeral source port in a benign communication.	
760	[**] MY.NET.30.4 activity [**] Quite a number of external systems connecting to ports 80, 524 and 51443 on this system. 51443 is the secondary HTTPS port (next to 443) used by Novell Netware Enterprise Server, which ties in nicely with the presence of port 524, which is Netware Core Protocol (NCP). Consequently, this appears to be a Novell Netware box which is being played with by external users.	As configured
629	[**] Queso fingerprint [**]	Not analyzed
535	[**] IDS552/web-iis_IIS ISAPI Overflow ida nosize [**] Not analyzed further (see below for ISAPI INTERNAL rule)	Not analyzed
492	[**] Tiny Fragments - Possible Hostile Activity [**]	Not analyzed
447	[**] Possible trojan server activity [**] Rule triggers on all communication involving the SubSeven trojan port 27374. The mayority of the alerts are the result of one external system, 61.73.196.198, scanning portions of the MYNET.edu network for this particular port. Most other alerts are false positives triggered by the use of port 27374 as ephemeral source port in a communcation. No evidence of an actual SubSeven backdoor on MYNET.edu.	Low
432	[**] connect to 515 from outside [**]	Not analyzed
335	[**] EXPLOIT x86 NOOP [**] Rule matching on a sled of x86 NOP instructions (0x90). Without having copies of the packets which triggered this alert, nothing much can be said about whether these alerts are false positives or the real thing. Since most of the trafficappears to be responses from external web and news servers to internal clients, we suspect that the rule is triggering frequently on benigng payloads.	Low
283	[**] TCP SMTP Source Port traffic [**] All 283 alerts are the result of this rule triggering on 128.220.43.220:25 hammering away on MY.NET.204.74:27 for about ten seconds.	Low
280	[**] NETBIOS NT NULL session [**] Three internal systems, MY.NET.137.34, MY.NET.137.46 and MY.NET.190.100 seem to incite the curiosity of two external systems on 12.28.135.133 and 12.35.158.199	Medium
214	[**] External RPC call [**] Rule seems to trigger on every external connection attempt to port 111 on an internal system.	As configured
178	[**] MY.NET.30.3 activity [**] Ports 80, 524, 1433, 2200 and 8009 on this system seem to incite a certain external interest. System appears to be a Novell Netware Server.	As configured
175	[**] CS WEBSERVER - external ftp traffic [**] IDS rule seems to catch and report all external connections to	

	this web/ftp server.	
148	[**] IRC evil - running XDCC [**] DCC is a file sharing feature of IRC (Internet Relay Chat). XDCC bots are commonly used to "offer" storage capacity for "warez" on hacked servers or servers with open file shares. XDCC is also used to announce the names of available warez files on IRC channels. The 19 (nineteen) distinct MYNET.edu systems triggering this alert could be offering pirated software or audio/video using University resources as repository. A more detailed analyisis on this alert follows below.	**High**
89	[**] TFTP - External TCP connection to internal tftp server [**]	Not analyzed
89	[**] NMAP TCP ping! [**]	Not analyzed
78	[**] EXPLOIT x86 setuid 0 [**] Rule matching on |b017 cd80| in the byte stream. Without having copies of the packets which triggered this alert, nothing much can be said about whether these alerts are false positives or the real thing. From the variety of destinations and ports involved, I suspect the former.	Low
53	[**] EXPLOIT x86 stealth noop [**]	Not analyzed
36	[**] IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL [**] Internal hosts launching ISAPI attacks. 11 distinct internal systems were caught by this rule, with MY.NET.98.102 and MY.NET.98.47 being the most prominent with 12 / 6 attacks against them. The listed sort alert was formerly part of the standard Snort rulebase and triggered on the string ".ida?" in the packet payload. Since this pattern is rare enough in an average data stream, MYNET.edu should investigate the systems detected by this rule more closely. The detected activity is not sustained enough, though, to conclude that the systems caught by this rule are being abused for an all-out ISAPI attack or have been infected by a worm doing the selfsame thing.	**High**
28	[**] EXPLOIT x86 setgid 0 [**]	Not analyzed
26	[**] SNMP public access [**] Alerts were triggered by a three external systems which tried SNMP access with a "public" community string against a number of internal systems. Since the IDS rule seems to trigger on the stimulus packet (the request), there is no telling whether the targeted systems did respond or not.	Medium
26	[**] Notify Brian B. 3.54 tcp [**] Rule seems to trigger whenever system MY.NET.3.54 is contacted by an external host. Nothing overly untoward is apparent from the 26 logged connection attempts.	As configured
17	[**] TFTP - Internal UDP connection to external tftp server [**]	Not analyzed
17	[**] Notify Brian B. 3.56 tcp [**] Rule seems to trigger whenever system MY.NET.3.54 is contacted by an external host. Nothing overly untoward is apparent from the logged connection attempts.	As configured
17	[**] Attempted Sun RPC high port access [**] Rule triggered only once, on an access attempt from 205.188.153.97:4000 to MY.NET.209.90:32771. Since this rule	Medium

	triggers only 17 times (compared to the 5778 alerts caused by the "SUNRPC highport access" alert rule listed further up), we conclude that this rule is also matching on packet payload. Hence, these 17 alerts might well be "the real thing" and indeed constitute evidence of an attempted Sun RPC attack. No other log entries on either source or destination could be found in the logs, though.	
6	[**] Probable NMAP fingerprint attempt [**]	Not analyzed
6	[**] Port 55850 udp - Possible myserver activity ref. 010313 [**] The alerts are caused by MY.NET.140.9 port 55850 talking to one external system, 130.18.27.33, on six different high ports. From this port pattern, it looks as if the external system has indeed been contacting the MYNET.edu system on the "myserver" backdoor port. No other evidence was found in any of the logs (OOS, scans).	Medium
5	[**] FTP passwd attempt [**] Rule possibly triggering on an attempt to retrieve /etc/passwd from an FTP server. Rule is firing on the stimulus (request), no telling whether the request was successful.	Low
4	[**] SMB C access [**] Attempt to connect to shared drive C: on a Windows system. Systems targeted are MY.NET.132.43 (twice), MY.NET.190.94 and MY.NET.190.100	Low
2	[**] RFB - Possible WinVNC - 010708-1 [**] Rule seems to trigger on the use of port 5900, which is also employed by WinVNC, a Windows remote control program. Internal systems MY.NET.162.91 and MY.NET.84.187 both triggered this alert once.	Low
2	[**] PHF attempt [**]	Not analyzed

XDCC

Source Address	# Entries	Alert(s) Logged	Severity
MY.NET.114.142	32	[**] IRC evil - running XDCC [**]	**high**
MY.NET.83.205	20		
MY.NET.91.151	15		
MY.NET.83.3	15		
MY.NET.237.106	12		
MY.NET.211.98	12		
MY.NET.162.91	12		
MY.NET.240.234	6		
MY.NET.217.42	6		
MY.NET.210.222	5		
MY.NET.222.106	3		
MY.NET.251.146	2		
MY.NET.207.6	2		
MY.NET.88.163	1		
MY.NET.84.250	1		
MY.NET.240.206	1		
MY.NET.227.106	1		
MY.NET.218.46	1		
MY.NET.116.103	1		

XDCC is a small program which frequently gets installed on compromised (hacked) systems. The program then actively joins Internet Relay Chat (IRC) channels to offer the compromised system as storage space for so-called "warez" (illegally shared copyrighted audio/video data or programs). Most common XDCC clients seem to also contain a "Backdoor" with administrative privileges. Without any evidence to the contrary, we conclude parties unknown are in total control of the systems listed above.

The standard Snort IDS rule base does not contain any specific rules for catching XDCC traffic. We therefore assume that the rule has been added manually by MYNET.edu staff and is likely triggering on "tell-tale" XDCC signs in the data stream. The most common signatures seem to be the strings "xdcc list" and "xdcc send" in an IRC data stream. A very good paper by "TonikGin" titled "XDCC - an .EDU Admin's Nightmare" [27] is available from http://www.russonline.net/tonikgin/EduHacking.html
Ample proof that MYNET.edu is not the only University battling the XDCC infestation and a very good write-up on countermeasures is available from Duke University [28].

Step Four - Locating sources of intensive scanning activity

This is the first step where we introduce the scan logs into the analysis process. Again, we start with a simple tally of the most frequent scanners, and also list the particular destination port the scanner was going after. This serves to both identify systems or users who "misbehave" and to get a quick impression of the services which are most "sought after". By far most of the scanners for TCP services were using a SYN scan (97.8%), with NULL (0.5%) and FIN (0.2%) scans being distant second. The table shown below is split into inside and outside scan originators to make the difference in services scanned for more apparent.

Scanners on the MYNET.edu network	
# Entries	Source and Dst Port
111186	MY.NET.223.78 -> 443
19731	MY.NET.70.176 -> 6257
10364	MY.NET.87.44 -> 27005
4071	MY.NET.97.110 -> 137
3480	MY.NET.97.136 -> 22321
3096	MY.NET.98.31 -> 22321
3073	MY.NET.98.31 -> 7674
2717	MY.NET.97.67 -> 22321
2634	MY.NET.98.150 -> 22321
2432	MY.NET.97.85 -> 137

Scanners on the outside	
# Entries	Source and Dst Port
5368	66.134.226.37 -> 443
2422	64.156.31.70 -> 80
1814	63.78.224.166 -> 80
1670	210.178.9.1 -> 443
1517	206.167.165.56 -> 443
1252	61.242.90.229 -> 80
802	213.73.142.100 -> 139
720	218.155.10.85 -> 22
702	12.239.36.3 -> 1433
581	195.6.68.65 -> 21

Table 4 - Top ten inside and outside scanners, together with the port scanned for

Most of the ports hunted for by external parties are as expected and as frequently seen in firewall and IDS logs around the globe. The ports scanned for by internal sources are more intiguing, though, and reveal some activity which we haven't picked up on in previous analysis steps.

Source Addr	# Entries	Reason	Severity
MY.NET.223.78	111186	Portscan of external systems for tcp/443	**high**

This internal system is running a heavy scan against several external networks, ranging from 217.41.* up to 217.80. The networks thus scanned belong to several European internet providers. The scan lasts for eleven hours and is noisy enough to almost ensure that it has been picked up by providers and IDS systems all over Europe. MYNET.edu staff should investigate if the system originating this scan has been compromised, or whether a student or employee is abusing University resources to attack foreign web sites.

Source Addr	# Entries	Reason	Severity
MY.NET.70.176	19731	Portscan of external systems for udp/6257	medium

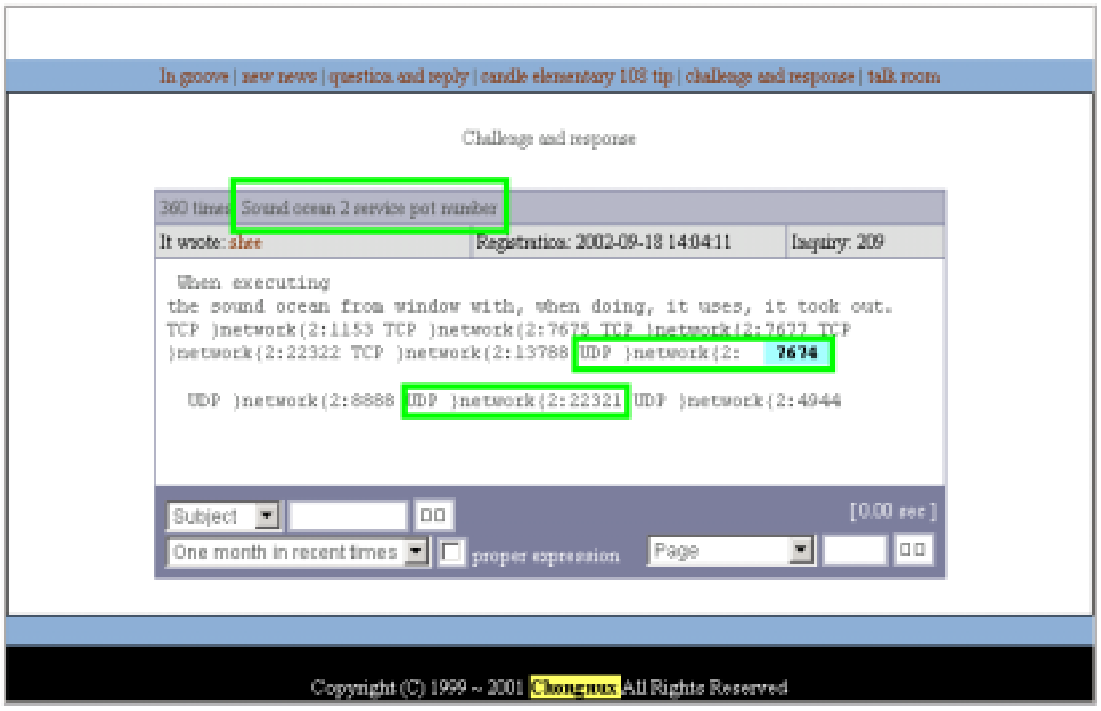
This internal system is apparently partaking in a WinMX peer-to-peer file sharing network and is actively looking for other WinMX hosts. The system has already been identified as a WinMX host in an earlier analysis step (see above).

Source Addr	# Entries	Reason	Severity
MY.NET.87.44	10364	Portscan of external systems for udp/27005	medium

From the logs, it appears as if the system's source port 27021 is scanning 191 distinct external systems for port 27005/udp. Clients of an internet-enabled multiuser game called "Half-Life" are using 27005 as udp source port when connecting to a Half-Life server [29]. The standard server port for HalfLife is 27015, though, and not 27021. Still, this activity could suggest that MY.NET.87.44 is running a HalfLife Server and that the seemingly "scanned" systems are in fact HalfLife Clients.

Source Addr	# Entries	Reason	Severity
MY.NET.97.136	5377	Portscan of external systems for udp/7674 and udp/22321	medium

From the logs, it appears as if the system's source port 7674 and 22321 is scanning 5153 distinct external addresses for port 7674/udp and 22321/udp respectively. The only place where Google could locate these two ports together on one page was on a Korean language website,
http://chongnux.klug.or.kr/board/read.php?table=hack&no=505
The page is also available through the Google cache should the Korean server be unreachable (search for chongnux 7674). Translated with Babelfish. the gist of the page (shown below) suggests that a tool or game called "Sound Ocean 2" is using these ports (among others) to communicate.



In groove | new news | question and reply | candle elementary 108 tip | challenge and response | talk room

Challenge and response

360 times Sound ocean 2 service pot number

It wrote: shee | Registration: 2002-09-18 14:04:11 | Inquiry: 209

```
 When executing
the sound ocean from window with, when doing, it uses, it took out.
TCP )network(2:1153 TCP )network(2:7675 TCP )network(2:7677 TCP
)network(2:22322 TCP )network(2:13788 UDP )network(2:    7674

  UDP )network(2:8888 UDP )network(2:22321 UDP )network(2:4944
```

Copyright (C) 1999 ~ 2001 Chongnux All Rights Reserved

Figure 1 Babelfish translation of Korean "SoundOcean" web page

Step Five - Identifying sources of Out-of-Spec data

As a final step of the data grinding routine, we take a closer look at those packets caught by the "out of spec" sensor. One apparent characteristic that stands out from the data is that systems on the MYNET.edu network seem to be mainly on the "receiving end" of mangled packets. The six MYNET.edu systems listed as originators of OOS traffic are the only ones which show up in five days' worth of OOS log.

# Entries	Destination of OOS Packet
463	MY.NET.70.225:4662
449	MY.NET.207.2:6011
447	MY.NET.220.106:4662
373	MY.NET.237.66:4662
368	MY.NET.229.58:3676
367	MY.NET.202.50:6346
325	MY.NET.6.47:25
319	MY.NET.24.21:25
….	…. …..
260	MY.NET.211.106:6346
194	MY.NET.233.10:4662
164	MY.NET.249.134:1214

#Entries	Source of OOS Packet
445	148.64.169.5
368	148.63.130.172
347	65.214.38.10
300	68.164.35.154
203	213.98.16.183
….	…. …..
53	MY.NET.12.4
15	MY.NET.12.2
3	MY.NET.252.14
3	MY.NET.244.58
1	MY.NET.253.2
1	MY.NET.238.86

*Table 5 - Top ten destinations and sources of "out-of-spec" traffic. The dotted lines indicate where a number of similar and repeating records have been removed. The source list has been adapted to show *all* inside sources of OOS traffic.*

Packet Length Mismatch

There is no "natural" cause for a packet header length mismatch, short of poorly written software or operating system errors. Nevertheless, quite a lot of packets end up in the OOS logs just because this error condition. Some EDonkey2000 file sharing clients seem to be especially prone to mangle packets. From the list above, we have already identified the systems MY.NET.237.66 and 220.106 before as likely EDonkey hosts. In addition to these, and thanks to mangled packets, the OOS logs now identify MY.NET.70.225 and MY.NET.223.10 as other likely EDonkey servers.

```
02/15-02:35:46.436162 148.64.169.5:4730 -> MY.NET.70.225:4662
TCP TTL:116 TOS:0x0 ID:49875 IpLen:20 DgmLen:48 DF
****P*** Seq: 0x84223E0A  Ack: 0x0  Win: 0x2000  TcpLen: 20
E3 37 00 00 00 01 10 28
```

The packet above shows an example of the length mismatch. Ordinarily, IPLength and TCPLength should add up to DgmLength, which is clearly not the case here. Two other quite uncommon characteristics of this packet are the odd "Push-Only" flags and the acknowledgement number set to zero.

Early Congestion Notification (ECN)

Quite a number of OOS packets seem to get caught by a rule matching on the two high-order (reserved/ECN) TCP flag bits set to "1". Nowadays, this flag combination is nothing much out of the ordinary anymore - it only means that the system sending the initial "SYN" of a new connection is capable of using the Early Congestion Notification (RFC 3168) mechanism.

```
02/15-01:09:47.642958 12.232.181.246:63001 -> MY.NET.207.2:6011
TCP TTL:48 TOS:0x0 ID:15715 IpLen:20 DgmLen:52 DF
12****S* Seq: 0xF75F17E3  Ack: 0x0  Win: 0x16D0  TcpLen: 32
TCP Options (6) => MSS: 1460 NOP NOP SackOK NOP WS: 0
```

Nevertheless, the logs also caught a few uncommon combinations of ECN flags. Both MY.NET.12.4 and MY.NET.12.2 were logged to emit TCP "R"eset packets having both ECN tcp flag bits set to "1", which is uncommon since the ECN flags are normally cleared in a reset response. This can either mean that the two MYNET.edu end systems themselves are not ECN compliant, or that the two systems are protected by a firewall product which is not ECN aware. This error condition is listed specifically in RFC3360, which deals with "inappropriate resets".

```
02/15-02:15:06.605288 MY.NET.12.4:143 -> 12.222.108.45:1942
TCP TTL:255 TOS:0x0 ID:39346 IpLen:20 DgmLen:40
12***R** Seq: 0x78E01006  Ack: 0x0  Win: 0x0  TcpLen: 20
```

Defensive Recommendations

While we are aware that tight network security control in an open academic environment is almost impossible, we still see several avenues to improve the defensive stance of the MYNET.edu network. Some of our recommendations focus on technical details, but we expect the biggest potential to lie in procedural changes.

A network based intrusion detection system, as installed at MYNET.edu, works by detecting anomalous or malicious traffic on the wire. In order to productively use an IDS, two boundary conditions must be met. First, it must be clearly defined what is considered to be "malicious" traffic at the site running the IDS. Secondly, an IDS should be backed up by filtering devices (firewalls) to ensure that traffic considered to be "malicious" does not reach the protected network in the first place. Without the first pre-requisite, neither the IDS administrators nor the users at the site in will know right from wrong. Without the second pre-requisite, the IDS analyst will be swamped by allegedly hostile traffic and will likely end up missing a real incident in all the clutter.

From the MYNET.edu log files which we have analyzed in the course of this engagement, we conclude that both pre-requisites are only partially met. For the IDS installation at MYNET.edu to be of any lasting effect, we therefore recommend MYNET.edu management to carefully consider the steps outlined below.

Define an "acceptable use" policy for MYNET.edu

The management should urgently take steps to ensure that the acceptable uses of the University's network are properly agreed on and documented. Students and staff should be required to acknowledge receipt and understanding of this policy document when joining the University. As a minimum, the document should prohibit hacking activity by University users and regulate / prohibit the use of P2P file sharing tools for the exchange of copyrighted materials. The document should also clearly outline the extent to which University staff has the right to monitor activity on University networks. Due to the sensitive nature of this policy document, it should be verified with subject matter experts from both MYNET.edu Information Security and Legal/Compliance departments before being put into effect.

Improve perimeter protection measures

Even in an University environment, not all connection requests inbound from the Internet have to be passed through. Likely, the University network could be segregated into three zones, one hosting the productive external services of MYNET.edu (like the public website), one containing the lab and administration networks, and one containing all the networks of the student housing complexes. Filtering for these three zones could then be gradually implemented from strict (for the external services) to moderate (for student housing). Even student dormitory networks need not be able to host server services on arbitrary ports - it should therefore be verified whether these networks can be firewalled off for inbound connections from the Internet, while leaving outbound connectivity unrestricted.

Improve IDS configuration and use

A sustained rate of more than thousand alerts per hour cannot reasonably be processed, not by automated tools and certainly not by an Intrusion Analyst sitting at the console. By defining an acceptable use policy and by restricting unwanted traffic from entering the University network (see above), the two most important pre-requisites to make the IDS system more worthwhile are met. As a next step, the IDS installed at MYNET.edu should urgently be upgraded and/or redesigned to reflect the current state of IDS practice. Much of the activity analyzed in the course of this engagement has been logged almost "by chance", caught by IDS rules which were inprecise enough to trigger on traffic other than what they were intended for. To avoid this, we recommend to start from scratch with a standard Snort rulebase, to cut it down to a ruleset reflecting the acceptable use policy, and then to fine-tune it to eliminate noise and false positives. Alert rules triggering more than a handful of times per hour should either be turned off or re-written. Once the alert rate has been lowered to an acceptable level, the IDS operation mode should be changed to capture packet logs of ALL packets triggering an alert. In order to do any meaningful analysis, and Intrusion Analyst needs to be able to see both the rule *and* the packet which caused the alert.

We also recommend to run specific IDS filters every now and then, to also get an impression of those portions of the University's network traffic which are not violating any of the standard IDS rules. A good example for such a filter would be one matching on "SYN-ACK" originating from student networks and leaving the campus, to easily locate "clandestine" servers on the internal network. This is of course not a reasonable IDS filter for productive use, but should be employed every now and then to monitor network usage (if tighter firewalling is not an option).

Data Reduction and Analysis Techniques

Log file cleanup and preparation

To start the processing, we first concatenated the five days worth of logs for each type (alert, scans, OOS) into one file each. While the date on the alert and scan file names matches the date of the content, the OOS files are apparently "off by one". Hence, to cover the five days between February 15 and 19, 2003, the following files were processed:

Date	***Alert File***	***Scans File***	***OOS File***
15/02/2003	alert.030215.gz	scans.030215.gz	OOS_Report_2003_02_16_32309
16/02/2003	alert.030216.gz	scans.030216.gz	OOS_Report_2003_02_17_6137
17/02/2003	alert.030217.gz	scans.030217.gz	OOS_Report_2003_02_18_27913
18/02/2003	alert.030218.gz	scans.030218.gz	OOS_Report_2003_02_19_479
19/02/2003	alert.030219.gz	scans.030219.gz	OOS_Report_2003_02_20_28598

The resulting concatenated files "all_alert", "all_scans" and "all_oos" were then analyzed further. A quick comparison between the "alert" and "scans" log revealed that all of the portscan alerts logged in the former are also listed , with more detail, in the latter. Consequently, it was possible to remove the log entries added to the alert file by the portscan preprocessor without losing information. This was done as follows

```
creosote:/home/giac # grep -v '(spp_portscan|Null scan!)'
all_alert.ori > all_alert
```

During this step, it became also apparent that the alerts file contained roughly 100 log lines which do not conform the the specification, i.E. do not start with a timestamp at the beginning of the line. This could be a result of the sanitizing process, which seems not to function properly at any given time. In order to facilitate further processing, these offending log lines were removed from the raw alert data as well. Unlike the OOS and alerts file, the scan log still contained raw, "unsanitized" IP addresses of the inside network. To facilitate comparisons between the three files, we have decided to also "convert" the addresses of the scans file into the "MY.NET" notation.

```
creosote:/home/giac # perl -pi.bak -e 's/130\.85/MY\.NET/g' all_scans
```

After these modifications to the raw data (removing the scan log entries and broken log lines from alerts file and sanitizing the IP addresses in the scan log), it was time to "deep-freeze" the logs to make sure that we could detect possibly inadvertent changes to the logs during the further analysis. To that end, the md5 checksums of the concatenated logs were taken and copied to a safe location.

```
creosote:/home/giac # md5sum all_oos all_scans all_alert
9c007679cc37b907f552ba891dfe0867  all_oos
d70c74d2961d748b8f7f00351efc8e8f  all_scans
14d52f6432230cafff7a15b0f9947767  all_alert
```

Compiling Timeline Statistics

As a next step, we put together a small Perl script to count the number of log lines for each date and hour. This script (see below) applied to all three logs, resulted in three lists of hourly activity (excerpt shown below) which was then imported into a spreadsheet to plot the timeline graph in the introduction chapter of this document.

```
#!/usr/bin/perl
while (<>) {
        # This matches on the date format used in the scans files
        if (/^\w{3}\s(\d\d\s\d\d)/) {
                $count{$1}++;
                next;
        }
        # This matches on the date format used in alert and OOS files
        if (/^\d\d\/(\d\d\-\d\d):\d\d:\d\d/) {
                $count{$1}++;
                next;
        }
        #remember to comment out the line below when processing the OOS file
        print "OUT OF SPEC: $_";
}

foreach $hour (sort {$a cmp $b} (keys %count)) {
        my ($d,$h)=$hour=~/(\d\d)[\s\-](\d\d)/;
        print "Feb $d @ $h,$count{$hour}\n";
}
```

```
Feb 19 @ 07,1855
Feb 19 @ 08,1983
Feb 19 @ 09,2022
Feb 19 @ 10,1944
Feb 19 @ 11,4201
Feb 19 @ 12,1461
Feb 19 @ 13,1608
Feb 19 @ 14,1246
Feb 19 @ 15,821
```

Table 6 Sample output of the timeline analysis script, formatted for easy importing into Excel

Extracting Addresses and Ports from the Alerts File

The first analysis step, aimed to extract systems on the inside network with "server-like" behaviour, was conducted at shell level with a one line combination of Perl and shell expressions.

```
creosote:/home/giac # perl -ne 's/\]\s(MY\.NET[\.\d]*):(\d*)\s/print
"$2 $1:$2\n"/e' all_alert | sort -n | uniq -c
```

This expression extracts all inside sources and source ports from the combined alert log file, sorts them according to source port number, and then counts the number of similar lines. An excerpt from the resulting output is shown below. The results were then imported into a spreadsheet for further analysis (like eliminating all entries with less than 30 occurences)

```
      3 0 MY.NET.204.94:0
     11 20 MY.NET.162.67:20
      2 25 MY.NET.6.35:25
      3 25 MY.NET.6.47:25
      1 69 MY.NET.100.225:69
    265 69 MY.NET.111.219:69
    275 69 MY.NET.111.230:69
    332 69 MY.NET.111.231:69
    313 69 MY.NET.111.232:69
    308 69 MY.NET.111.235:69
```

Table 7 - Frequency count of inside source ports

A similar script was used to extract those internal systems which were frequently contacted on a specific destination port. The alerts caused by the resulting about seventy internal systems were then extracted from the alerts file and investigated manually with search expressions in the "vi" editor. From the resulting thirty systems which were showing definite server like behaviour, the "most interesting" ones were picked for the analysis. "Most interesting" in this context can roughly be defined as "providing a learing opportunity for the analyst", since we decided early on to ditch the routine stuff like SMB access and to concentrate on the more "exotic" data.

Another bunch of Perl expression was used to extract the top alert sources, destinations, source ports and destination ports, as used in the second step of the analysis. All expressions basically consisted of

- a pattern to extract the data of interest from the alerts file
- a shell expression like "| sort | uniq -c | sort -rn" to sum up the results and to list them with the most frequent entry show first

```
creosote:/home/giac # perl -pe 's/.*\]\s//; s/\s-.*//; s/:.*//'
all_alert | sort | uniq -c | sort -rn  > top_alert_sources.txt
```

The command sequence shown above was used to tally the source addresses of systems causing alerts, the results of which were then compared against a list of the most frequent *combinations* of source address and alert message. The latter information was extracted with the command shown below.

```
creosote:/home/giac # perl -ne 's/\[\*\*\]\s(.*)\[\*\*\]\s([^:\s]*)/
print "$2 $1\n"/e' all_alert | sort | uniq -c | sort -rn | head -10
```

```
 13180 MY.NET.211.6 Incomplete Packet Fragments Discarded
  4722 169.232.84.146 SUNRPC highport access!
  2184 212.179.123.163 Watchlist 000220 IL-ISDNNET-990517
  1753 12.35.158.199 SMB Name Wildcard
  1280 212.179.88.96 Watchlist 000220 IL-ISDNNET-990517
  1154 212.179.105.210 Watchlist 000220 IL-ISDNNET-990517
  1018 MY.NET.132.42 Incomplete Packet Fragments Discarded
   964 212.179.91.129 Watchlist 000220 IL-ISDNNET-990517
   805 141.157.254.236 CS WEBSERVER - external web traffic
   770 MY.NET.207.214 High port 65535 tcp - possible Red Worm
```

Table 8 - Sources with a particular high number of the same alert message

From this analysis step, it became apparent that in the context of MYNET.edu IDS logs, the frequency of an alert is a very poor indicator of the criticality of the event. The most frequent alerts were obviously those which were caught by deliberately added, but very broad rules like the one used to catch traffic originating from a certain network in Israel ("Watchlist 000220").

As a next step, the alerts themselves were tallied up, using the same filtering expression as above, but dropping the tie to the source address. The unexpected result was that the huge number of alerts contained in the combined alerts log apparently only consisted of 44 individual alert messages, with the most frequent one ("SMB Name Wildcard") occurring almost 75'000 times. The low number of distinct alert messages made it possible to manually analyze almost all of them as part of this third analysis step.

In the fourth round, the scan log was analyzed to extract the top ten scanners and ports hunted for. As before, a combination of a Perl filtering expression and shell-level sorting was employed to get the desired results. The command used to extract the top inside scanners together with the port hunted for is listed below.

```
creosote:/home/giac # perl -ne 's/(MY\.NET[\d\.]*).*-
\>\s\S*:(\d*)/print "$1 -> $2\n"/e' all_scans | sort | uniq -c | sort
-rn | head -10 > top_inside_scanners
```

The resulting list was then verified manually and checked against the other two log files to produce the analysis listed in the previous chapter. A similar filtering expression was then used to parse the top sources and destinations of data logged in the "Out of Spec" (OOS) log file. Due to the more complex nature of the OOS log file, where packet headers are mixed with entire packet dumps, most of the actual analysis on the OOS data was performed manually.

References

Part One - Describe the State of Intrusion Detection

[1] Textor, Steve: "Installation and Configuration of a Cisco PIX Firewall" April 29,2002 http://www.sans.org/rr/firewall/cisco_pix.php
[2] Setting up PIX Syslog, http://www.cisco.com/warp/public/110/pixsyslog.html
[3] Cisco PIX Firewall System Log Messages, Version 6.2 http://www.cisco.com/univercd/cc/td/doc/product/iaabu/pix/pix_62/syslog/index.html
[4] Monitoring Cisco PIX Firewall with Syslog through a VPN Tunnel http://www.cisco.com/warp/public/110/pix_vpn_4094.html
[5] Wilson, Curt: "Cisco PIX attack patterns research", November 3, 2000 http://www.sans.org/y2k/110300.htm
[6] Bird, Tina and Ranum, Marcus: Log Analysis Resources http://www.loganalysis.org
[7] Cisco PIX Messages Listed by Severity Level http://www.cisco.com/univercd/cc/td/doc/product/iaabu/pix/pix_62/syslog/pixemapa.htm
[8] Cisco PIX Command Reference, Version 6.2 http://www.cisco.com/univercd/cc/td/doc/product/iaabu/pix/pix_62/cmdref/

Part Two - Practical Detects

[9] IETF RFC 2236, IGMP http://www.ietf.org/rfc/rfc2236.txt
[10] IETF RFC 3376, IGMPv2 http://www.ietf.org/rfc/rfc3376.txt
[11] Securiteam.com - IGMP Denial of Service http://www.securiteam.com/securitynews/5XP0B1F7FY.html
[12] Coan, Brian et al, IGMP Security Problem Statement and Requirements http://www.securemulticast.org/GSEC/gsec3_ietf53_SecureIGMP1.pdf
[13] NSFocus.com Security Advisory - Netbios Password Verification http://www.nsfocus.com/english/homepage/sa_05.htm
[14] F-Secure.com - Opasoft Worm Description https://www.europe.f-secure.com/v-descs/opasoft.shtml
[15] CERT Advisory CA-2002-36 Multiple Vulnerabilities in SSH http://www.cert.org/advisories/CA-2002-36.html
[16] Provos, Niels and Honeyman, Peter, ScanSSH http://www.citi.umich.edu/techreports/reports/citi-tr-01-13.pdf
[17] Miller, Toby, Passive OS Fingerprinting Details and Techniques http://www.incidents.org/papers/OSfingerprinting.php

Part Three - Analyze This

[18] Roesch, Martin and Green, Chris, Snort Documentation http://www.snort.org/docs
[19] F-Secure.com Information on RedWorm/Adore Backdoor http://www.f-secure.com/v-descs/adore.shtml
[20] KaZaA.com "KaZaA Media Desktop" http://www.kazaa.com/us/index.php
[21] EDonkey2000.com "What ports does EDonkey use" http://www.edonkey2000.com/cgi-bin/smartfaq/smartfaq.cgi?answer=1025114514

[22] WinMX.com "WinMX - The best way to share your media"
http://www.winmx.com/
[23] Gnutella.com Website
http://www.gnutella.com/
[24] Sun Microsystems Security Bulletin 142: Vulnerability in portmapper
http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=secbull/142
[25] Securiteam.com Vulnerability Alert - Spike Denial of Service
http://www.securiteam.com/windowsntfocus/6G00B2K5PM.html
[26] IANA Assigned Port Numbers List
http://www.iana.org/assignments/port-numbers
[27] TonikGin - "XDCC, an .EDU admin's nightmare"
http://www.russonline.net/tonikgin/EduHacking.html
[28] Duke University OIT Security Team, "Instructions on cleaning XDCC"
http://www.oit.duke.edu/security/cleaning/xdcc.html
[29] Scarborough, Matt, "Signs of Internet Gaming"
http://www.incidents.org/detect/gaming.html

Other tools and Websites frequently used in the course of this analysis

- DShield.org Distributed NIDS database
  http://www.dshield.org
- MyNetWatchMan.com Distributed NIDS database
  http://www.mynetwatchman.com
- Incidents.org Mailing List Archive
  http://cert.uni-stuttgart.de/archive/intrusions
- Demon.net "Network Tools" web page
  http://www.demon.net/external
- Switch.ch "Whois Query Tools"
  http://www.switch.ch/search/whois_form.html
- Snort Rulebase Documentation
  http://www.snort.org/cgi-bin/sigs-search.cgi
- Google, the Mother of all Search Engines
  http://www.google.ch