Crying wolf: Of false positives and irrelevant attacks.

GIAC Certified Intrusion Analyst (GCIA)
Practical Assignment
Version 3.3 (revised August 19, 2002)

Knut Bjørnstad

September 22, 2003

Later, he saw a REAL wolf prowling about his flock. Alarmed, he leaped to his feet and sang out as loudly as he could, "Wolf! Wolf!" But the villagers thought he was trying to fool them again, and so they didn't come. At sunset, everyone wondered why the shepherd boy hadn't returned to the village with their sheep. They went up the hill to find the boy. They found him weeping.

"There really was a wolf here! The flock has scattered! I cried out, "Wolf!" Why didn't you come?"

Aesop's Fables

Contents

Part 1. Describe the state of intrusion detection: False positives and irrelevant attacks

Introduction

My idea for Part 1 was to take a look at the situation when someone decides to deploy Snort, without having an idea about tuning the rule set. I think the process from the start of those new to IDS has been a bit overlooked. A bit strange since all new intrusion analysts must have been at this stage once.

Neither Beale et al. "Snort 2.0 Intrusion Detection"[1 and 2] or the Snort documentation has much to say about tuning. I did not find much mention of this when searching with Google either. But when I fire up Snort 2.0 with the default rule set, I get nearly half a million alerts, almost all without interest.

A consequence of this is that you have to be a specialist to have any use of NIDS at all. It also follows that NIDS will be expensive, since those who pay cannot use ordinary network administrators for this task. I don't propose to solve this fully here, what I do is a practical review of the first stages in a tuning process - what to discard of the rules and why in my own firms home network. This isn't meant to be a tutorial for the removal of false positives either. To make that I think much more work should be done to make it understandable for the beginner. Besides, this is written much more from the perspective of a service provider, than e.g. a bank where things should be simpler in some ways. So I think this isn't general enough as a tutorial.

Goal: To have fewer than 150 alerts a day - and at the same time not losing too much in the form of false negatives.

Note about myself: I am in an intermediate position here - I am in no way new to NIDS , having worked on NIDS log analysis for ca. 3 years. But this I have done in a second line position, while other people have produced the logs. Then I have checked on the events in the logs in our fairly complicated network, communicated with the server people and so on. So I have had little direct experience with the NIDS tools themselves (we have used NFR, RealSecure and Snort). It has in this way been a learning experience for me to work on Snort in connection with this practical.

Source of the observations

As mentioned this is based on logs from my own firm's network. We are a Norwegian services firm, and are fairly large in a Scandinavian context (but far from the size of the ones in the big nations and on the international scene). We have a mix of servers network that we have control over (and where I have access to the servers and can check logs), and customer networks where we have to have their consent to check on security problems. This also means that choice of policy is dependent of business conditions, and will be quite complex. For a conceptual sketch of how the NIDS sees the network see Figure 2 in Detect #3, part 2.

We also have lots of firewalls in our network, sometimes in several layers. As we have a large address range (around a million addresses), we see lots of the random scanning and sweeping traffic on the network.

We have our production NIDS sensors outside our border router. They are placed on a TopLayer switch, which has a port I can use for experimental purposes. I have put a Linux box on this switch port for the analysis I have made here.

A few words on the methods I have used to analyse the logs in this practical. Instead of making a complicated script, or downloading such a script from the network I have used one line commands or short scripts, mixing perl, shell and awk. I have not listed all the commands/scripts I have used, but mentions the most interesting ones in the text.

I have used the Snort fast log format, analysing the logs in ASCII format.
Here is an example of how I extract addresses and makes a count ranked for descending frequency:

The script xtr_ipad:
```
#!/bin/sh
perl -ne 'if (/ (\w+\.\w+\.\d+\.\d+) -> (\w+\.\w+\.\d+\.\d+)/){print "$1
$2\n"} elsif (/ (\w+\.\w+\.\d+\.\d+):\d+ -> (\w+\.\w+\.\d+\.\d+):\d+/)
{print "$1 $2\n"};
if (/ (\w+\.\w+\.\d+\.\d+).\d+ > (\w+\.\w+\.\d+\.\d+).\d+/ ){print "$1
$2\n"} ' $*
```

Used in a command to count IP-adresses:

```
xtr_ipad |sort|uniq -c|sort -nr
```

This is not elegant or effective maybe, but it works. Besides I have cut and pasted data back and forth, and edited them if necessary. I have used the same methods trough the whole practical.

My idea is to take a look at what happens when someone downloads snort and starts sniffing with default signatures and plugins. If your network have a certain size and complexity what you experience might be compared to opening a closet door and being drowned by what is inside. Of course this action might also be seen as the very first stage in a tuning process, like it is in my case.

But I suspect that in many cases someone with a reasonable competence on Linux or Windows servers fire up Snort (or some other NIDS tool) and get drowned in this avalanche of alerts. It is easy enough to do this - it is much worse to understand the details of what they

see. What to do next might in many cases be:
1. Give up and decide IDS is not for them and decide to close their eyes and hope all attackers will overlook them.
2. Even worse, they denounce someone as an attacker based on the false positives.

I was involved with one example of this when a firm complained of being under attack by a certain person. When I looked at the logs they sent me (this was actually from a personal firewall, not an IDS sensor), I saw at once that this was DNS answer packets that the logs reported as a port scan. But why did they then denounce the person in question? It appeared they made a whois request on the DNS server address. The record showed the person as responsible for the address (he was not any longer, but the record had not been changed yet), so they sent him a mail demanding that he should stop port scanning them. The person, being a bit arrogant just answered that port scanning is not illegal, so shut up. So then they had proof he was the attacker . . . . .

Stage 1: First time deployment of Snort

Now I install Snort-2.0.0, set it up to listen on the interface connected to the TopLayer switch and let it run for 24 hours, keeping all the defaults - except telling what our DNS servers and SMTP servers are. Note that the port scan pre-processors are commented away in the default Snort 2.0.0 installation, so I get no alerts on port scans here.

No	#of alerts	Message	SID
1	89261	MS-SQL Worm propagation	2003
2	55462	ICMP Destination Unreachable(Communication Administratively Forbidden)	485
3	35563	SNMP request udp	1417
4	35070	SCAN nmap TCP	628
5	21614	SCAN Proxy (8080)attempt	620
6	13177	ATTACK RESPONSES 403 Forbidden	1201
7	9623	WEB-CGI adcycle access	1721
8	7437	ICMP Large ICMP	499
9	5753	(snort_decoder): T/TCP Detected	56
10	5714	WEB-CLIENT javascript URL	1841
11	4803	ICMP Source Quench	448
12	4746	SNMP public access	1411
13	2318	WEB-PHP content-disposition	1425
14	2016	ICMP L3retriever Ping	466
15	1399	MISC Tiny Fragments	522
16	1358	WEB-FRONTPAGE /_vti_bin/ access	1288
17	1206	WEB-IIS view source via translate header	1042
18	922	WEB-MISC http directory traversal	1113
19	740	WEB-CGI redirect access	1443
20	632	WEB-CGI scriptalias access	873
21	531	WEB-CGI calendar access	882
22	459	WEB-CGI search.cgi access	1599
23	382	WEB-MISC /doc/ access	1650
24	357	WEB-MISC login.htm access	1564
25	320	WEB-FRONTPAGE posting	939
26	309	WEB-MISC robots.txt access	1852
27	306	WEB-CGI count.cgi access	1149
28	304	WEB-IIS fpcount access	1013
29	266	DDOS shaft client	230
30	258	DDOS mstream client	247/

			249
	308115	Total number of alerts	

Table 1. 30 most frequent alerts from first 24 hour run. Note that the Snort decoder alerts have no signatures but are hard coded in the program. They have a SID, but they are not listed on www.snort.org's SID documentation pages.

Evaluation of what to take away

Here I try to evaluate some of the most frequent alerts in order to get to a manageable noise level. When I say delete rule I mean comment away the one-line rule in the Snort rule file. This is done both when false positives is the cause of the alerts, and when the traffic causing the alerts are seen as irrelevant noise attacks that does not threat us enough to be alerted. Note: When I mention retaining logs as info, I mean classifying them to a least critical category, which is read by the IDS analyst only when a threshold is exceeded, or as correlation for other events. This might be done by a post-processing script, or by Snort's *severity* option, which might be filtered on afterwards. However I do not comment much on post-processing here. The references here are - as can be seen [2-5]

Row 1) SID 2003 - MS-SQL Worm propagation attempt
Reference: http://vil.nai.com/vil/content/v_99992.htm, http://www.securityfocus.com/bid/5311 and 5310
Analysis: This of course is no false positive, but a sad reflection of the level of applied security on the Internet. The population of unpatched servers on the network seems to be more or less constant. This may also reflect that some people just reboot their servers when it (and the LAN it is on) is infected. Since this worm only resides in memory, they will probably have a period of function before they are re-infected.

For us this is a very small threat. If someone in our network get infected in spite on our explicit policy against letting SQL Server listen on the Internet we will get warned at once. There is another rule, SID 2004 "MS-SQL Worm propagation attempt OUTBOUND" which reports infected machines on your own network. If we should get infected, it has to be on an internal network via some backdoor, but there we have other pretty good control routines, which I think will prevent a worm from affecting us too much. I classify this as an irrelevant attack.
Action: Delete rule

Row 2) SID 485 - ICMP Destination Unreachable (Communication Administratively Prohibited)
Reference: Just http://www.snort.org/snort-db/sid.html?sid=485
Analysis: Ca. 75% of this are from our outer routers - this must clearly be filtered away. It is however hard to specify just their few (2 - 10) addresses and show the rest because of limited address format in *snort.conf*. Some of this traffic might mean that someone is spoofing our addresses to probe others. This happens all the time with our large address range, but in our context this is must be considered background noise. The other part of the traffic is due to various addressing errors. So this is a mix of false positives and irrelevant attacks.
Action: Delete rule

Row 3) SID 1417 - SNMP request udp

Reference: http://www.snort.org/snort-db/sid.html?sid=1417
Analysis: Company policy is to stop incoming SNMP at the border firewalls, so this rule might be deleted. The high number of alerts here is due to two internal servers communicating with customer devices outside our network. The default `snort.conf` definitions EXTERNAL_NET and INTERNAL_NET as any are the reason these are seen here. Outgoing SNMP traffic should also be restricted (to a few addresses having business doing SNMP) to avoid attacks on others from our home network. This might be reflected in snort.conf with a variable OUTGOING_SNMP_ADR containing the addresses seen as normal here.
Action: Redefine network variables. Possibly delete if noise level is too high, but a post-processing script giving informational counts would be better.

Row 4) SID 628 - SCAN nmap TCP
Reference: http://www.whitehats.com/info/IDS28
Analysis: The great number of alerts here is not false positives, but reflects an agreed security test
Action: Retain rule as is

Row 5) SID 620 - SCAN Proxy (8080) attempt
Reference: http://www.snort.org/snort-db/sid.html?sid=620
Analysis: There clearly must be errors in the Snort documentation page. The reference to FTP etc makes no sense in my view and is also repeated in the SID 618 docpages. The rule is:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 8080 (msg:"SCAN Proxy \(8080\)
attempt"; flags:S; classtype:attempted-recon; sid:620; rev:2;)
```

This merely alerts on all SYN packets to port 8080, which is a popular alternative to port 80, besides being used by web proxies. These alerts are in all probability 100 % false positives. Note that I made a posting to the *snort-sigs* mailing list, reporting the documentation error.
Action: Delete rule

Row 6) SID 1201 - ATTACK RESPONSES 403 Forbidden
Reference: http://www.snort.org/snort-db/sid.html?sid=1201
Analysis: This response are logged in web access logs, so should be unnecessary. It would be best to create some sort of alert from these logs, perhaps starting when some threshold has been passed. But this lies in the host based IDS realm.
Action: Delete rule

Row 7) SID 1721 - WEB-CGI adcycle access
Reference: http://www.snort.org/snort-db/sid.html?sid=1721, http://www.securityfocus.com/bid/3741
Analysis: This CGI signature is undocumented on www.snort.org, but Securityfocus says: " AdCycle is a set of shareware ad management scripts written in Perl and back-ended by MySQL". It has an input validation error. Generally WEB-CGI signatures are hopeless generators of false positives, since all these CGI's are used in actual URL's on some web servers. Besides lots of websites (ours and others) don't use them any more. I treat all CGI signatures the same way as this, so I don't mention the other SID's (1433, 873, 882 and1149).
Action: Delete or retain logs as info. Might be upgraded to a higher severity for vulnerable sites on our home network, perhaps making a *CGI_WEBSERVERS* variable in *snort.conf*

Row 8) SID 499 - ICMP Large ICMP
Reference: http://www.snort.org/snort-db/sid.html?sid=499

Analysis: This is meant to warn against DOS attacks, but most modern stacks would be immune I think (but I have not had the time to check this thoroughly). The Snort doc page says "A number of load balancing applications use 1500 byte ICMP packets to determine the most efficent route to a host by measuring the latency of multiple paths. ". This means that most or all alerts are false positives.
Action: Delete rule

Row 9) SID 56 - (snort_decoder): T/TCP Detected
Reference: not documented at all at www.snort.org
Analysis: For references and analysis of this see the section *Detect #1*
Action: Delete rule. Note: since this is no proper rule but part of the decoder, uncomment *config disable_ttcp_alerts* in *snort.conf* to do this.

Row 10) SID 1841 - WEB-CLIENT javascript URL host spoofing attempt
Reference: http://www.securityfocus.com/bid/5293
Analysis: The reference: "(It is) possible to create a javascript: URL which appears to start with a valid domain. Malicious script code may specify an arbitrary domain, and will be able to access cookie data associated with that domain". The signature is "javascript://" which seems too general - though my knowledge of this language is very limited. The vulnerability concerns Mozilla up to 1.0. Most users use other browsers, and the few who do will hopefully use newer versions. The high noise level - presumably due to ordinary web pages with this string - and low risk lead to a delete decision.
Action: Delete rule

Row 11) SID 448 - ICMP Source Quench
Reference: Undocumented at www.snort.org (except the signature code where the message says Unknown code! but this is not in the distributed code)
Analysis: This merely alerts on all source quench packets (Type 4 code 0). This might possibly be used for probing or a DOS attack, but should be blocked at the borders, so this attack is not very important. Source Quench is used for flow control, alerting a sender when packets are coming to fast. A comment by W. Richard Stevens [10], page 161 indicate that this ICMP message was seen as obsolete already in 1993, so we should not need it. Actually, a common opinion is that only one ICMP message should be let through from the outside - Type 3 code 4 "Fragmentation needed but don't fragment bit set". This is necessary to allow path MTU discovery, more on this in the comment to row 15.
Action: Delete rule

Row 12) SID 1411 - SNMP public access udp
Reference: Just http://www.snort.org/snort-db/sid.html?sid=1411
Analysis: This merely alerts on the string "public" in a UDP 161 packet incoming to the home network. Like most default Snort rules it is dependent on only one packet, so a scan on all addresses for this will generate as many alerts. Incoming SNMP should have been blocked in border firewalls, so should be no great threat. Outgoing SNMP should also be restricted, or at least controlled by the NIDS, see comment to row 3. The default community string "public" (The SNMP v. 1 "password")should of course never be used, since it makes information leak attacks very easy. SNMP v. 1 is all over a very insecure protocol that should be restricted as much as possible.
Action: Delete rule or retain log for information.

Row 13) SID 1425 - WEB-PHP content-disposition

Reference: http://www.securityfocus.com/bid/4183
Analysis: Note: www.snort.org says this is disabled by default - seems to be wrong.Signature is content:"Content-Disposition\:"; content:"form-data\;"; Securityfocus does not reveal the PHP-exploit, but packets I have looked into does not seem to carry anything sinister. Furthermore this signature does not adress PHP specifically, and will have false positives with non-PHP requests.
Action: Filter away outgoing requests.

Row 14) SID 466 - ICMP L3retriever Ping
Reference: http://www.whitehats.com/info/IDS311
Analysis: The reference says: "This event may indicate that someone is scanning your network using the L3 "Retriever 1.5" security scanner....
"This type of ICMP ping seems to be also generated by (plain) Win2K host talking to Win2K domain controllers." --nnposter ". I have no idea why this is so common, but the firewalls should block incoming pings of all kinds.
Action: Delete rule

Row 15) SID 522 - MISC Tiny Fragments
Reference: Undocumented at www.snort.org
Analysis: Clearly this rule is meant to discover fragmentation attacks. Lots of false positives are generated by VPN's in our network. IPsec based VPN's has a special problem, because the protocol adds an extra header to the packets, often making them longer than the maximum path MTU. This will then lead to small fragments corresponding to this addition. Often, when the Don't fragment bit is turned on to make path MTU discovery this will generate problems with the communication. But this rule will generate false positives regardless of there being such problems or not.
Action: Filter away VPN gateway (or adjust MTU size of the packets).

Row 16) SID 1288 - WEB-FRONTPAGE /_vti_bin/ access
Reference: http://www.securityfocus.com/bid/4251
Analysis: This records ordinary use of FrontPage - I assume this is harmless :-) But the last vulnerability was reported just a year ago - so we should be aware of scans and the like. Best would be to filter away a list of administrators and report all other access - if such a list could be made.
Action: Retain log as info or filter away admin addresses.

Row 17) SID 1042 - WEB-IIS view source via translate header
Reference: http://www.securityfocus.com/bid/1578
Analysis: Signature is "Translate|3a| F" - since this is a normal web option (with a explicit ":" not the hex code) it will generate lots of false positives. With us a large part of the alerts are on non-IIS-servers.
Action: Filter away non-IIS-servers, for IIS: retain logs as info

Row 18) SID 1113 - WEB-MISC http directory traversal
Reference: http://www.whitehats.com/info/IDS297
Analysis: Signature is "../" , this was tested against one website in our network - it had this string in 5651 places in its web directories! I did not check if all files were in active use however. There is also a signature SID 1112 which seems to have a bug, it seems to trigger on

packets without the signature, which is "..\\ ". The threat here is relatively mild, so I tend to put the burden on those responsible for the web servers.
Action: Delete rule

Rows 19 -22 and 27: WEB-CGI sigs - see the comment on row 7

Row 23) SID 1560 - WEB-MISC /doc/ access
Reference: http://www.securityfocus.com/bid/318/discussion/
Analysis: This is Debian Linux specific. We use this distribution very little or not at all, so this can be deleted. If it will be used more, it will be a newer version, which is patched for this error.
Action: Delete rule

Row 24) SID 1564 - WEB-MISC login.htm access
Reference: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-1533
Analysis: The reference is about a specific attack on an Eicon modem. But more generally login.htm authentication is unsafe, so one might protect the web serves in the home network with this rule. But in my firm this gets too noisy, so I take it away. Besides, this should be in the web logs.
Action: Delete rule

Row 25) SID 939 - WEB-FRONTPAGE posting
Reference: http://www.snort.org/snort-db/sid.html?sid=939
Analysis: See also the comment on row 16. This will record all Frontpage access. This might be better done in the web logs, but if one wants, one might have a list of addresses with legitimate access, and filter on this.
Action: Delete rule

Row 26) SID 1852 - WEB-MISC robots.txt access
Reference: http://cgi.nessus.org/plugins/dump.php3?id=10302
Analysis: Most web servers in our network do not use robots.txt. We must assume that with those who do, there must have been an evaluation of the dangers of information leak. This signature is noisy mainly because there are some broken search machines on the network.
Action: Delete rule

Row 28) SID 1013 - WEB-IIS fpcount access
Reference: http://www.securityfocus.com/bid/2252
Analysis: The reference says "A vulnerability in the package could allow a user to execute arbitrary code on a running server." The rule is:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS
fpcount access";flow:to_server,established; uricontent:"/fpcount.exe";
nocase; reference:bugtraq,2252; classtype:web-application-activity;
sid:1013; rev:6;)
```

This alerts here on outgoing traffic and of all use of this program. Better would be to define the `$HTTP_SERVERS and $HTTP_PORTS` variables to restrict this to incoming traffic. In our setting it is hard to get a working list of web servers, but it can be done with a systematic scan on the most used ports. This could lead to a list of several hundred addresses though. A better value of `$EXTERNAL_NET` will help also.
Action: Change variables in *snort.conf*

Row 29) SID 230 - DDOS shaft client
Reference: http://www.whitehats.com/info/IDS254
Analysis: This fire on the high port number 20432, so you get a count of how often this are used in e.g. web return packets (some.outside.web.server.80 -> my.net.xx.yy. 20432). In my experience DDOS clients are very rare, I have never caught one in network of my firm. This rule will not help, as the real thing will drown in the noise. Sven Dietrich [56] has an analysis of this DDOS tool - there are commands from the client that could be used to make a content string.
Action: Delete rule- should be rewritten.

Row 30) SID 247 and 249 - DDOS mstream client
Reference: http://www.snort.org/snort-db/sid.html?sid=247
Analysis: The two signatures have the same message, so is taken together here. SID 247 reacts to *content: ">"; flow:to_server,established;* and port 12754. I checked out three packets, the first two contained obvious web data (Verisign certificates) and the third was an ACK package, where the > was part of the datagram length field (hex 3c)! This is not good enough, how can anyone hope an ">" will not occur in ordinary traffic to port 12754? This also points out an important failing of signature based NIDS - it is weak in keeping state - even with the *flow* option . This will lead to false positives on return traffic from the server.

The snort rule page surprisingly claims there are no known positives with this rule. Evauating how to catch this DDOS attack falls outside the scope of my simple analysis, but according to Dave Dittrich [57] there are client commands that could be used in a signature *content* option together with the port number.
Action: Delete rule - should be rewritten.
SID 249 - DDOS mstream client
Reference: http://www.snort.org/snort-db/sid.html?sid=249
Analysis: This rule reacts to destination 15104 and reserved bits 1 and 2 set. False positives should happen more rarely than with the former SID. Her 7 of the 258 alerts had this SID.
Action: Retain rule

Stage two - taking away the worst signatures and enabling port scan

After trying the default rule set, the following actions was taken:

$HOME_NET was updated to reflect the actual network instead of *any*, using a list of CIDR network specifications. *$EXTERNAL_NET* is hard to specify as anything other than *any*, but with a little help, I at last specified it as a list of 36 CIDR-notation network addresses!

It might be better to use MAC addresses to specify the two networks.

SID's #1841, *1852*, *1113, 882* and *2003* vas deleted
Now I have deleted more than I specify in the tables with increased risk of false negatives.

Snort does not seem to have a way to subtract networks from a variable in *snort.conf*. I tried a not (!) operator in one variable $IR_EKSTERN to filter away some ICMP Host unreachable - this did not work, so I deleted SID 485 instead.

Then I uncommented in /etc/snort/snort.conf:

```
preprocessor portscan: $HOME_NET 4 3 portscan.log
preprocessor portscan-ignorehosts: my.net.1.2@53 my.net.1.4@53
my.net.2.10@53 my.net.2.20@53
```

The result now was:

No	#of alerts	Message	SID
1	4310	SNMP public access	1411
2	2789	(snort_decoder): T/TCP Detected	56
3	1958	spp_portscan: portscan status	2
4	543	ICMP Large ICMP	499
5	337	ICMP L3retriever Ping	466
6	330	WEB-FRONTPAGE /_vti_bin/ access	1288
7	234	spp_portscan: PORTSCAN DETECTED	1
8	234	spp_portscan: End of portscan	3
9	161	ICMP Destination Unreachable (Communication with Destination Network is Administratively Prohibited)	487
10	153	ICMP Source Quench	448
11	109	WEB-IIS view source via translate header	1042
12	88	SCAN Squid Proxy attempt	618
13	82	WEB-PHP content-disposition	1425
14	55	MISC Tiny Fragments	522
15	55	ICMP PING CyberKit 2.2 Windows	483
16	47	WEB-FRONTPAGE posting	939
17	44	ICMP PING speedera	480
18	34	WEB-MISC apache DOS	1156
19	29	(snort_decoder) WARNING: TCP Data Offset is less than 5!	46
20	27	WEB-FRONTPAGE shtml.dll access	940
21	26	SCAN nmap TCP	628
22	23	WEB-IIS _vti_inf access	990
23	19	DDOS shaft client	230
24	18	WEB-FRONTPAGE shtml.exe access	962
25	14	SCAN FIN	621
26	12	WEB-MISC whisker space splice attack	1104
27	10	BAD TRAFFIC tcp port 0 traffic	524
28	9	SMTP HELO overflow	1549
29	6	WEB-CGI scriptalias access	873
30	6	WEB-CGI redirect access	1443
	11797	Total number of alerts	

Table 2. 30 most frequent alerts from second stage run.

Now there are much fewer alerts - in all 11797, but still far from the goal of under 150 alerts a day. Some signatures we have not investigated now make the top 30: Rows 9, 12, 15-20, 22 and 24-28. I will not evaluate these alerts like I did with first set-up, in order to keep this analysis at a reasonable length. But this must be done much the same way.

And then we have the *spp_portscan* rows, and a *portscan.log* file. Now others may have a different opinion of this, but I don't feel that it is terribly important to be alerted on incoming port scans. A network installation like ours will be port scanned more or less continually, and following up this will use too many resources. And it is legal in most countries, as some people remind us of sometimes.

Besides I feel that the border firewalls is a better tool for logging port scans than a NIDS installation. So at the next stage I won't be logging port scans any more.

Stage three - a bit better

In addition to the SID's removed in stage 2 I now removed SID's 620, 1201, 1721, 499, 448, 1411, 1425, 466, 1560, 1564, 939 and 230

I did not do all the other actions specified in the comments to Stage one. The port scan pre-processor was removed again.

No	#of alerts	Message	SID
1	3083	(snort_decoder): T/TCP Detected	56
2	1586	SNMP request udp	1417
3	232	WEB-FRONTPAGE /_vti_bin/ access	1288
4	130	SCAN Squid Proxy attempt	618
5	87	ICMP Source Quench	448
6	70	ICMP Destination Unreachable (Communication with Destination Network is Administratively Prohibited)	487
7	70	ICMP Destination Unreachable (Communication with Destination Host is Administratively Prohibited)	486
8	60	ICMP PING CyberKit 2.2 Windows	483
9	58	WEB-IIS view source via translate header	1042
10	52	SCAN nmap TCP	628
11	46	MISC Tiny Fragments	522
12	46	ICMP PING speedera	480
13	28	WEB-PHP content-disposition	1425
14	18	WEB-IIS _vti_inf access	990
15	10	WEB-FRONTPAGE shtml.exe access	962
16	8	WEB-FRONTPAGE shtml.dll access	940
17	6	BAD TRAFFIC tcp port 0 traffic	524
18	6	SMTP HELO overflow	1549
19	4	(spp_stream4) STEALTH ACTIVITY (unknown) detection	1
20	3	ICMP PING NMAP	469
21	3	DDOS mstream client	247
22	3	ICMP PING NMAP	469
23	3	DNS zone transfer UDP	1948
24	2	(snort_decoder): Truncated Tcp Options	55
25	2	(snort_decoder) WARNING: TCP Header length exceeds packet length!	46
26	2	WEB-IIS _mem_bin access	1286
27	2	WEB-MISC apache DOS	1156
28	1	WEB-MISC Transfer-Encoding: chunked	1104
29	1	(snort_decoder): Tcp Options found with bad lengths	54
30	1	(snort_decoder) WARNING: TCP Data Offset is less than 5!	46

	5623	Total number of alerts	

Table 3. 30 most frequent alerts from third stage run.

I thought that a factor here might be that stage 3 was done a month further into the summer than the first all-defaults run, so the traffic volume might be lower. To check on this I made a new 24-hour run, but actually the number of alerts was higher - 458899 as compared to 308115 in the earlier run.

From the above analysis, the rules in rows 1 and 21 should have been deleted (in the last case I forgot). And now we see some very interesting alerts showing themselves at the bottom of the list - if we get too many false positives we get too little time for them. But that still leaves 2540 alerts, 17 times my goal of 150 alerts. By changing rule variables and tighten ICMP policy as described above, it might be possible to remove all or most of the alerts in row 2 and 5-8. Then we are down to 667 alerts - four times my goal. Of course this is much better than trying to digest nearly half a million alerts every day, but clearly still too much.

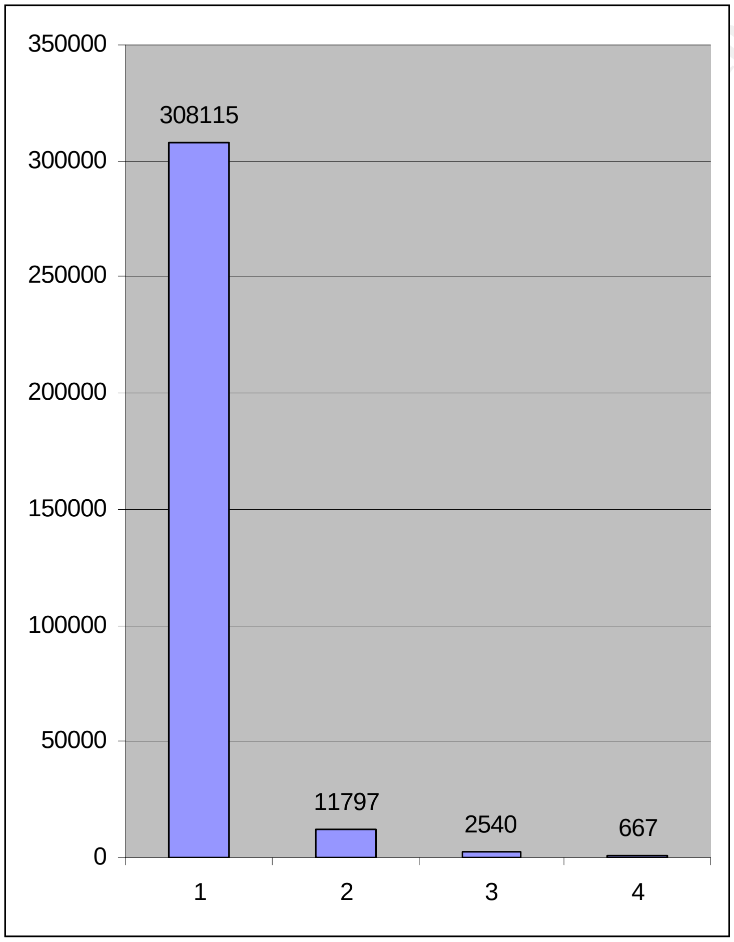

Figure 1. A bar chart that show gain in the three stages. The fourth bar are stage 3 with the removals described above.

The road further on - from simple to advanced tuning

I have clearly reached an end to my simple method of throwing away the most noisy and unproductive signatures and adjust a few configuration variables. So what should be done next? I stop my analysis here and just sketch some ideas.

- Rules should be revised, and new ones added. Its clear from my analysis that some of the Snort default rules are not good enough. Its not hard to observe that some are missing either - when using another NIDS tool in parallel like ISS RealSecure you will see some alerts that Snort misses (and the other way round too). But when rules are changed and added the thorny question of performance tuning pops up. A NIDS at a busy Internet access point will soon start to lose packets if you do something wrong.

Note: It might be thought that the commercial NIDS tools are better at avoiding false positives than Snort, but that is not my experience. When I looked at RealSecure a couple of years ago, the false alerts popped up in ten thousands, while their signature descriptions had the annoying tendency to claim "No false positives" for most of them.

- The next stage should undoubtedly also mean making a post processing script that can filter away more flexibly than Snort. This should perhaps be based upon Snort's ACID database interface. But be aware that there is a performance aspect to this - if we let Snort log massively, and filters afterwards, it may choke.

Some ideas for consideration by NIDS tool developers:
- Some method for testing for false negatives should be available. I don't know how this can be done, perhaps some kind of test battery. When you take away the noise you will lose information too - the question is how to minimalise this. Even my simple first stage stripping here are problematic in this respect.

- I wish someone could make some sort of question and answer tuning tool for Snort. This might make it easier to use for some people, and broaden the useful use of IDS. I am thinking of some kind of automation of the process above, where a test run is made, the results presented (including packet content in a easy to read form - perhaps using ethereal or something even better), changes written back to configuration files, and then repeating the process.

- I also wish better session analysis capacity. This should mean the end of the return traffic false positives that plagues us today. But it should also be possible to retain the whole session, so the tool, or the analyst can do a step-by-step analysis to determine what is going on in the session. This will consume huge amounts of disk and processor capacity, but I think it is possible.

- The reference [53] describes an interesting way of analysing IDS systems. Their Receiver operating characteristic curve, plotting detects against false alerts seems a good way of doing this. But as I have not emphasised detects here (there were few if any of interest during my runs actually), I will not go further into this.

References

I put them at the end of the practical, but I have separated the ones for this part and put them first.

Network Detects

Detect #1 - T/TCP traffic

1. Source of the trace

This detect was from the network of my own firm. Lately we have been seeing some "(snort_decoder): T/TCP Detected" SID 65 (no proper signature actually since it is made by the decoder). I thought I might try to find out the reason for this, even if there seems to be no evidence that this is an attack - see below.

My idea was that sorting out this traffic might help to find out if this is a false positive, or if it is some sort of attack. I try to analyse this in the next section. This was posted to *intrusions@ incidents.org* [17] and commented on by Andrew Rucker Jones [18] giving valuable advise.

Note that in the following I restrict the analysis strictly to the part of the network of my firm that I have full access to. We have some customers with web servers and network that I don't have full access to - there is a theoretical possibility that T/TCP might be deployed there (though in practice I find this highly unlikely - and we have never detected this either) In the part of the network I comment on here I know there is no T/TCP - I have inspected the servers.

2. Detect was generated by

We have set up an additional Snort 2.0.0 sensor for test and educational purposes. See part one on this sensor. For a conceptual graph of our IDS sensors, see Figure 2, Detect #3.

The Snort decoder has an annoying feature - changing all port numbers to zero when it discovers a TCP header anomaly. But the packet dumps showed that all the packets were SYN packets for port 80 or 443. All events related to existing web servers, there were nothing to suggest random probing or scanning. E.g. during 22 hours on July 3 we had 3359 alerts of this type (of a total of 18257) so this is a quite noisy type of event.

T/TCP is described by W. Richard Stevens in [10]. Among its characteristics is that it assigns a 32-bit connection count (CC) value to connections it opens. The CC.NEW option here is the CC value of the address initiating the connection. This value is monotonic in the mathematical sense - i.e. it grows for each packet. The server will accept packets from the client if the value is greater than that of the last packet. This connection mechanism is called TAO - TCP Accelerated Open.

The point here is to achieve acceleration by avoiding the three-way handshake and cutting down on packet overhead, and by shortening the TIME_WAIT delay. Mark Stacey [16] discusses the practical gain. There is also a good and concise treatment in the Phrack article [15]. If the destination does not respond, the source falls back to normal TCP. Since we have no T/TCP servers to my knowledge, this happens in all cases.

I also checked the source addresses from the 22 hours of log. They were all from the outside of our network. The whois records of the addresses showed that almost all were from

Scandinavian ISP's, firms and institutions. This corresponds to our normal traffic since we are a Norwegian service provider. If one checks through the entire month a few addresses dominates. Running the 38 addresses through DNS showed that most of those that had PTR RR's (26) had names that seemed to indicate that they were firewalls or servers rather than clients. (Names with gw (2), fw (2), mail (8), exchange (8), ns (3)).

I also tried passive fingerprinting of the traffic. Of the tools (p0f, disco and ettercap) I tried I only managed to get p0f to work. I only allocated limited time to this however. p0f came up with UNKNOWN for all except one address, this was identified as FreeBSD 4.3 - 4.4PRERELEASE. The packet logs seems to have too little information for p0f. For references to passive fingerprinting see [21-25].

A manual comparison with the table I found in http://isc.sans.org/diary.html (this reference has disappeared since I found it - see [25] instead) - an updated summary of Toby Millers paper about passive OS fingerprinting, also came up with nothing:

```
Most common (30), was:
window: 16384
ttl:~64
mss:present - varying
dont frag: set
Window scale: not present
sackok: unset
nop flag: set
declared packet size:52

The others (exept the Free BSD one), 7, was:
window: 16384
ttl:~64
mss:present - varying
dont frag: unset
Window scale: not present
sackok: unset
nop flag: set
decl packet size:68
```

These results were from July. In September there was released a new version of p0f , 2.0.1. I made a new packet log of the T/TCP traffic which was still there as before. But one of the

```
[kbjo@minestrone p0f]$ ./p0f -NUSls ../tcpd070903_ttcpere|grep
81.0.147.33|perl -ne '/- (.*) Signature:/; print "$1\n" ' 2>/dev/null
|sort|uniq -c |sort -nr|less
p0f - passive os fingerprinting utility, version 2.0.1
(C) M. Zalewski <lcamtuf@coredump.cx>, W. Stearns
<wstearns@pobox.com>
p0f: listening on '../tcpd070903_ttcpere', 160 fingerprints, rule:
'any'.
    140 Windows 2000 SP4, XP SP1 (2)
     36 Windows 95 (low TTL)
     29 Windows XP Pro SP1, 2000 SP3
     12 Cisco Content Engine
      3 Windows XP SP1 (2)
      1 Windows XP/2000 (RFC1323) [tos 9] [GENERIC]
      1 Windows XP/2000 (RFC1323) [tos 3] [GENERIC]
      1 Windows XP/2000 (RFC1323) [tos 10] [GENERIC]
      1 Windows XP (leak) (PLEASE REPORT) [GENERIC]
```

source addresses gave a slightly better result now. In the listing above I have separated and counted the OS field of the output. As can be seen, this illustrates my point that this might be some kind of gateway, modifying the headers somewhat, but retaining some of the parameters set by the OS stacks. This seems to fit roughly with what I see in the access logs. The other top sources weren't as good, giving UNKNOWN as result.

I mailed some of the address owners and asked them if they could tell me what this is, but got just one answer saying they didn't know. Informal talk with sources at the providers didn't know either, because this is on customers networks they don't access. Finally I contacted the owner of one of the source address gateways. They told me that this was an ordinary Checkpoint Firewall-1 with a NAT address for their clients. This is still puzzling, since I have found no reference connecting Firewall-1 with T/TCP.

3. Probability the address was spoofed

First a general note on address spoofing: By spoofing is meant the changing of the source address to that of an innocent third party in order to hide where a network packet comes from. It is of course hard to prove that this has been done directly - to do this it is necessary to have a trace of the progress of the packet through the network. Instead we have to analyse the motives behind the attack, sometimes supported by the knowledge of the likelihood that the attack really comes from the source address.

One class of attacks that has a high probability of address spoofing is DOS attacks, since the attacker has no need of seeing the return traffic. Another class is attacks that tries to execute a command blindly on the victim address. This is hard to do with TCP, because the attacker will need the ACK packets to update the sequence numbers. A third very elaborate attack method with spoofed source is when the attacker has inserted a mechanism in the return path that can redirect or record the return traffic.

As I conclude that the events here all are normal traffic through unconventionally configured devices, there is no chance of spoofing here. A SYN flood attack might use T/TCP initiation to make a bigger effect (see subsection 5 below), but there is no evidence of excessive SYN packets here.

4. Description of the attack

```
1152 MSIE 6.0 Windows NT 5.0
 533 MSIE 5.5 Windows NT 4.0
 386 MSIE 5.5 Windows NT 5.0
 350 MSIE 5.0 Windows NT
 137 MSIE 6.0 Windows NT 5.0
  82 MSIE 6.0 Windows NT 5.1
  51 MSIE 5.5 Windows NT 5.0
  37 MSIE 5.5 Windows NT 4.0
  16 MSIE 5.01 Windows 95
   8 MSIE 6.0 Win32
   8 MSIE 5.0 Windows 95 DigExt
   7 MSIE 6.0 Windows NT 5.2
   5 MSIE 6.0 Windows 98 Win
   3 MSIE 4.01 Windows 95
   1 MSIE 5.0 Windows 95
```

I have access to some of the web servers that was the destination in the alerts, and looked up a few of the addresses in the access logs (the investigated web servers was all Apache), and in all the cases I checked this seemed to be entirely normal web traffic. As noted all the packets was SYN's for port 80 and 443. The listing on the left is a count - in the access_log of two popular web addresses - of the browser identification filed for the four most active T/TCP addresses.

Now this was a very puzzling case - ordinary clients doing things like looking up the weather with old Windows and IE versions, and using something as exotic as T/TCP! Actually, when I asked the web experts in my firm they could barely remember having heard of this protocol. Andrew Rucker Jones said the same. Some of the documentation seems somewhat old - the home page [14] seems to be untouched since W.Richard Stevens died.

I still lack enough information to conclude what is really going on here. Actually I think it is significant hat the browser and OS identification varies. I think this suggests some network device, appliance or similar. This must then, in addition to NAT'ing the source address, add to the TCP options in order to convert to T/TCP. But this is alas just a theory, I have no certain proof, I am afraid. As mentioned above one of the addresses proved to be on an ordinary Checkpoint firewall with NAT.

The deployment of T/TCP might have been done on purpose, though I find it strange that we have not been informed of this. After all there has to be T/TCP servers to make this worthwhile. Another explanation is that this has happened accidentally - see *6. Correlations*.

Here is an example of one of the syn packets:

```
14:15:17.477629 rr.ss.tt.uu.44825 > ww.xx.yy.zz.http: S
3924543636:3924543636(0) win 16384 <mss 1460,nop,wscale
0,nop,nop,timestamp 351677 0,nop,nop,ccnew 2662999>
0x0000   4500 0044 eba5 0000 3b06 bdb9 c159 8145        E..D....;....Y.E
0x0010   8b75 0841 af19 0050 e9eb c894 0000 0000        .u.A...P........
0x0020   c002 4000 a577 0000 0204 05b4 0103 0300        ..@..w..........
0x0030   0101 080a 0005 5dbd 0000 0000 0101 0c06        ......].........
0x0040   0028 a257
```

I have deleted both addresses here since all evidence hitherto points to no wrongdoing.

Here is the TCP option part of an Ethereal analysis of a packet:

```
Options: (28 bytes)
         Maximum segment size: 1460 bytes
         NOP
         Window scale: 0 bytes
         NOP
         NOP
         Time stamp: tsval 351676, tsecr 0
         NOP
         NOP
         CC.NEW: 2662982
```

All evidence shows this as not being an attack - if it was the attack must be masked as ordinary web requests. But I see no reason to do this, since we do not use T/TCP and would be unaffected by this. Theoretically someone might fish for vulnerable T/TCP servers and hide it among ordinary traffic, but I found no evidence of that.

5 Attack mechanism

Since there is no attack here, there is no attack mechanism. The alerts cannot be described as false positives, as the traffic seen is actually T/TCP SYN's. Most people would classify them as irrelevant alerts I think.

Instead I will describe how T/TCP might be attacked here. The main problem is session hijacking, which is the theme of the Phrack article [15].

Since the CC variable is used to decide if a packet is acceptable, it is quite easy to start a connection with spoofed source. Just choose a CC value that is in the higher part of the range, and there is a good chance that the server will accept it and insert it into an established session. Besides, probably all valid packets from the trusted address will be rejected, since their CC value will usually be lower.It is possible to perform attacks like the classical Mitnick attack described in "Network Intrusion Detection"[9], chapter 7. The victim here has to have a trust relationship with other machines based on IP address, like in the R commands. You can then send a SYN with source spoofed to a trusted host and with data part containing an attack command - this is allowed in T/TCP. This attack is now much easier, because there is no need to predict the sequence number. Actually this will set up a new session, so there is no need of insertion. But even when there is a session validated with a password, a similar attack might succeed as described.

But T/TCP is, as the first T in the name says (Transaction TCP) intended for web traffic and equivalent. In almost all cases you have other mechanisms for authentication and session control - a web session usually consists of several TCP sessions. The TCP connections are often accepted from any address - so IP authentication is not relevant here. When you use encryption, like SSL or SSH this keeps a crypto session for you. But this is a session in an entirely different meaning - it is in the application layer, and really independent of the transport layer. These protocols might be attacked also, in a man-in-the middle attack for instance. But that is outside the scope of this treatment, I think.

So if you use T/TCP carefully you might avoid these problems. But access control based on IP address on a web server might be problematic, even if a web server usually has no privileged access.

The Phrack article also mentions a possibility of SYN flooding being made worse by T/TCP because of queuing of the data in failed TAO packets (ordinary SYN packets should have no data). Another problem is that SYN cookies cannot be used if one deploys T/TCP. In conclusion the Phrack article says that T/TCP is flawed because of its security problems. This might explain that it is not used more - but I don't feel qualified to evaluate its usefulness otherwise.

6. Correlations

I found very little mention of T/TCP in security forums on the Internet, at least when using google.
This however is interesting [42]:

"13.2 FreeBSD T/TCP bugs
We have found that with FreeBSD-2.2.2-RELEASE, there (are) some bugs with T/TCP.
+FreeBSD will try to use T/TCP if you've enabled the ``TCP Extensions.'' To
+disable T/TCP, use sysinstall and disable TCP Extensions, or add this to your
+/etc/rc files:

```
 sysctl -w net.inet.tcp.rfc1644=0"
```

This seems to suggest that much of the source of the T/TCP traffic might be misconfigured FreeBSD Squid proxies. But version 2.2.2 is from 1998, so it seems strange that this should be the case. In addition to T/TCP support by FreeBSD, there is patches for the Linux 2.4.2 kernel [49], and it is supported in SunOS (a very old version though - 4. 1.3) I have found no reference for Windows support however. For sources on information on T/TCP see above. The formal treatment is of course the RFC [13]

7. Evidence of active targeting

I found no such evidence. Since everything indicates this is normal traffic with some unconventional configuration, one cannot call this active targeting, even if the web traffic is targeted at the web servers of course.

8. Severity

Criticality: This depends on what we should call the targets - say these are the web servers themselves. Then I put this at 4, the web servers in question are critical, but there are several of them, with a service switch ensuring redundancy. The web servers are front ends (or reverse proxies if you will), and the more critical application servers are isolated behind a firewall. An attack on them should be put at 5. I find the value here depending on targeting actually, if an attack was directed at all web servers together, or at the servers behind this will influence the rating. But of course when like here, there is there is no targeting at all, I find this value most uncertain.
Lethality: I put this at one - a session hijack is relatively difficult to perform. But since there are no T/TCP, and no attack, there is nothing to hijack.
System Countermeasures: 5 since no T/TCP should be pretty absolute!
Network Countermeasures: 5 since nothing is needed
(4+1) - (5+5)= -5 and we could feel quite well. If we deploy T/TCP, we would get a less favourable result if we don't do more active countermeasures.

9. Defensive recommendation

As long as T/TCP is unused in a network, no defence should be necessary. Then the Snort behaviour can be disabled. If it is deployed however, the Phrack article might suggest that some form of alert might be turned on, preferably tightly filtered in order to see only the involved addresses. The protocol should be strictly restricted to web traffic and other small transaction types of traffic, with non-TCP session control. IDS and firewall rules (if this is possible with existing firewalls) should be adjusted to enforce this.

10. Multiple choice test question

Here is an example of a T/TCP SYN packet.

```
10:52:57.734943 rr.ss.tt.uu.3603 > vv.ww.xx.yy.http: S
2413535141:2413535141(0) win 16384 <mss 1460,nop,wscale 0,nop,nop,timestamp
3096524 0,nop,nop,ccnew 31630770>
0x0000   4500 0044 aa2b 0000 3d06 f1ec d944 7497        E..D.+..=....Dt.
0x0010   8b75 084b 0e13 0050 8fdb 9fa5 0000 0000        .u.K...P........
0x0020   c002 4000 d6e8 0000 0204 05b4 0103 0300        ..@.............
0x0030   0101 080a 002f 3fcc 0000 0000 0101 0c06        ...../?.........
0x0040   01e2 a5b2
```

What shows that this is a valid attempt to initiate a T/TCP session?
a) The IP protocol field (byte 9)
b) The timestamp TCP-option (byte 50-59)
c) The SYN flag in the TCP header (byte 33)
d) The CCNEW TCP option (byte 62-67)
e) Both d) and c)

The correct answer should be e)

Detect #2: Strange RST Ack port 16 to port 0 packets

1. Source of the trace

The detect are from the *http://www.incidents.org/logs/Raw* files. I decided on 6 strange packets from the log file *2002.5.25.* This detect was posted on *intrusions@incidents.org* [26]. Nobody commented, it seemed to drown in the rush of other people posting their detects. As has been noted by several others (e.g. Les Gordon's practical [40]) the *Raw* files are logged by a Snort installation with an unknown rule set. This limits analysis because response to stimuli, and other traffic between the involved addresses is not recorded.

2. Detect was generated by

I ran the file (actually all the Raw files) through a Snort-2.0.0 with a default rule set. Then I summarized the alert files (making a frequency count as mentioned in part 1), and selected the alert "(snort_decoder) WARNING: TCP Data Offset is less than 5!"

Then I listed all the packets involved by first generating a command file:

```
grep 'WARNING: TCP Data Offset is less than 5' */alert|perl -ne '@a=split
/ \[\*\*/; @b=split /:/,$a[0]; @c=split /\//,$b[0]; if
(/(\w+\.\w+\.\d+\.\d+):\d+ -> (\w+\.\w+\.\d+\.\d+):\d+/) {print "echo \"##
$c[0]: ##\";tcpdump -nvr $c[0] host $1\n"} '|sort|uniq >command_file
```

When executing this file I got 261 lines spread out over all the Raw files, with seemingly very little in common except the Data Offset abnormality. At last I selected the following:

```
16:43:27.094488 24.206.159.155.16 > 46.5.184.162.0: R [bad tcp cksum f9f9!]
0:8(8) ack 0 win 12450 [RST 4.0.....] (ttl 49, id 18609, len 40, bad cksum
a814!)
16:43:27.094488 24.206.159.155.16 > 46.5.184.162.0: R [bad tcp cksum f9f9!]
0:8(8) ack 1 win 12450 [RST 4.0.....] (ttl 49, id 18609, len 40, bad cksum
a814!)
16:43:30.104488 24.206.159.155.16 > 46.5.184.173.0: R [bad tcp cksum f9f9!]
0:8(8) ack 0 win 12450 [RST 4.0.....] (ttl 49, id 18610, len 40, bad cksum
a808!)
16:43:30.104488 24.206.159.155.16 > 46.5.184.173.0: R [bad tcp cksum f9f9!]
0:8(8) ack 1 win 12450 [RST 4.0.....] (ttl 49, id 18610, len 40, bad cksum
a808!)
16:43:36.074488 24.206.159.155.16 > 46.5.184.248.0: R [bad tcp cksum f9f9!]
0:8(8) ack 0 win 12450 [RST 4.0.....] (ttl 49, id 18611, len 40, bad cksum
a7bc!)
16:43:36.074488 24.206.159.155.16 > 46.5.184.248.0: R [bad tcp cksum f9f9!]
0:8(8) ack 1 win 12450 [RST 4.0.....] (ttl 49, id 18611, len 40, bad cksum
a7bc!)
```

Hex dump of the first packet:

```
16:43:27.094488 24.206.159.155.16 > 46.5.184.162.0: R [bad tcp cksum
f9f9!] 0:8(8) ack 0 win 12450 [RST 4.0.....] (ttl 49, id 18609, len 40,
bad cksum a814!)
0x0000   4500 0028 48b1 0000 3106 a814 18ce 9f9b        E..(H...1.......
0x0010   2e05 b8a2 0010 0000 0000 0000 0000 0000        ................
0x0020   3414 30a2 0214 0000 0000 0000 0000             4.0...........
```

Here is a verbose tethereal analysis of the same packet:

```
Frame 1 (60 bytes on wire, 60 bytes captured)
    Arrival Time: Jun 25, 2002 16:43:27.094488000
    Time delta from previous packet: 0.000000000 seconds
    Time relative to first packet: 0.000000000 seconds
    Frame Number: 1
    Packet Length: 60 bytes
    Capture Length: 60 bytes
Ethernet II, Src: 00:03:e3:d9:26:c0, Dst: 00:00:0c:04:b2:33
    Destination: 00:00:0c:04:b2:33 (Cisco_04:b2:33)
    Source: 00:03:e3:d9:26:c0 (Cisco_d9:26:c0)
    Type: IP (0x0800)
    Trailer: 000000000000
Internet Protocol, Src Addr: 24.206.159.155 (24.206.159.155), Dst Addr:
46.5.184.162 (46.5.184.162)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 40
    Identification: 0x48b1 (18609)
    Flags: 0x00
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 49
    Protocol: TCP (0x06)
    Header checksum: 0xa814 (incorrect, should be 0xa20e)
    Source: 24.206.159.155 (24.206.159.155)
    Destination: 46.5.184.162 (46.5.184.162)
Transmission Control Protocol, Src Port: 16 (16), Dst Port: 0 (0), Seq: 0
    Source port: 16 (16)
    Destination port: 0 (0)
    Sequence number: 0
    Header length: 12 bytes (bogus, must be at least 20)

0000  00 00 0c 04 b2 33 00 03 e3 d9 26 c0 08 00 45 00   .....3....&...E.
0010  00 28 48 b1 00 00 31 06 a8 14 18 ce 9f 9b 2e 05   .(H...1.........
0020  b8 a2 00 10 00 00 00 00 00 00 00 00 00 00 34 14   ..............4.
0030  30 a2 02 14 00 00 00 00 00 00 00 00               0...........
```

tethereal seems to give up on the TCP header because of the bogus length - see [20].

I make the following observations:

1) Both IP header checksum and TCP checksum are incorrect. This is probably a result of the rewriting of the home network addresses to make them unrecognizable, as has been remarked in several practicals. I will ignore this in the following.

2) Destination port 0 and source port 16 are both unassigned to any TCP service. Note that nmap rejects use of port 0 as outside the legal port range.

3) Bogus TCP header length 12 bytes - the rest of the packet - 6 bytes - are just zeroes, if one assumes the real header length are 20.

5) Sequence number 0 and acknowledgement number 8. If I don't misunderstand the algorithm for these two numbers, the last can be seen as a result of being sent a SYN packet with sequence number 0 and packet size 8. But tests show that the response (RST ACK) has 0:0 as result.

6) The packets are binary equal two and two, and the timestamps are the same. At first I thought there might be a doubling in the log file, but it does not seem so. I checked if all, or many of the records in the file were equal, but they weren't. This might mean that the same packet has been sent twice with a very short interval between them. Note however that the IP id fields are the same in the pairs. Normally this should mean that the pairs are duplicates, there might be some reason for Snort logging the same packet twice. On the other hand this field might also be crafted.

There is 3 seconds between the two first pairs, and 6 seconds between the next two pairs. I can give no exact explanation for this, but the same program must have been responsible as the interval is too short to be explained by individual human keying.

7) Bit 2 in the reserved field in the TCP header (numbered 1-6 from the left) is set.

So how could one explain these highly abnormal packets? One might think they were the result of a stimulus with abnormal header fields, so I set out to test this. RST ACK should be the normal response when trying a port that has no working service.

The whois record of 24.206.159.155 showed that this belongs to a network of Shaw Cablesystems US (Kingswood Cable) in Calgary, Canada. I found it not proper to contact them, since this is an event that occurred more than a year ago.

3. Probability the address was spoofed

Below in the *Attack mechanism* subsection I do a lab test to see what could have caused these strange events. The analysis there makes it unlikely that there was a stimulus to 24.206.159.155 with spoofed sources (i.e. the destination in the seen packets) because of the abnormalities of the packets. Then either the packets come from the address 24.206.159.155, or the packets have spoofed source - see next section. This depends on the motive behind the attack (or we might say event, there is no certainty that this is an attack), but we lack information to find out what this is.

If the source address is spoofed this might be some kind of DOS attack, but then there must be more traffic. I see no way the six packets can harm the destinations directly. The same argument can be used about a blind attack, since the packets have no payload. As for a spoofed attack with redirection - we lack information to suppose this - but it is possible.

4. Description of the attack

From my analysis above there might be the following possibilities
1) *24.206.159.155* might have been compromised, or someone with legal access uses it for probing our network.
2) It might be an NAT address, and someone behind this address is doing the same.
2) The source address is spoofed

In either case someone seems to have tested a tool for crafting packets. The aim might have been to do a scan of the home network of *2002.5.25*, by getting routers to send ICMP host unreachable for those addresses that are not in use. But I see no reason for modifying the header fields and provoking IDS alerts on that.

An alternative might be some sort of profiling since Windows and Linux reacts differently to the header length field. I have not found a reference to this, but some of the articles on profiling must have mentioned this.

I tested nmap's way of doing OS profiling - it seems to differ from what I see here. It does not use the header length field and it does not allow port 0. It does set bit 2 in the reserved field in the TCP header. Besides, it does lots of other tests that Snort ought to alert on. For a discussion of how nmap does profiling, see [45].

But there are lots of other profiling tools, most of whom I have not studied, but Queso appears to set both bits. If we assume that Snort does not have rules that react to the rest of the traffic this might be a good theory. The six packets seem to few to give a good profile. And then there might be someone just testing to see what happens, perhaps having mistyped the addresses and intending some address in their own network (If you have addresses that is one key press away from a private network, you will see this a lot)

I must also mention that there is a trojan named Skun which operates on Windows, that uses port 16 among several others. But I don't think it is this, both because of the platform, and because it seem just an ordinary trojan without the special features mentioned here. See [28] for a list of trojan ports, and [29] for a somewhat superficial description.

5 Attack mechanism: Simulating the packets in my own network

At last it might be a trojan somewhat like Q (se Les Gordon's practical [40]). But then what we have must be just a fragment of the traffic, since it seems to carry very little information. It might be some kind of wake up signal perhaps.There is very scant information here that I can build on directly. In order to find out something about the behavior here I decided on a lab test. First I tried hping2 - my.net.12.50 is a Linux RedHat 8.0 box, and my.net.13.14 is a Win 2000 box.

```
hping2  --count 1 --baseport 0 --destport 22 --setseq 0 --syn  --data 8
my.net.12.50
hping2  --count 1 --baseport 0 --destport 16 --setseq 0 --syn  --data 8
my.net.12.50
hping2 - --count 1 --baseport 0 --destport 22 --setseq 0 --syn --tcpoff 3
--data 8 my.net.13.14

tcpdump -nv host my.net.12.50 or host my.net.13.14
...
11:09:00.324025 eth0 > my.net.2.233.0 > my.net.12.50.ssh: S 0:8(8) win
512 (ttl 64, id 28606)
11:09:00.325209 eth0 < my.net.12.50.ssh > my.net.2.233.0: S
291769782:291769782(0) ack 1 win 5840 <mss 1460> (DF) (ttl 62, id 0)
11:09:00.325270 eth0 > my.net.2.233.0 > my.net.12.50.ssh: R 1:1(0) win 0
(ttl 255, id 21181)
11:09:04.517830 eth0 < my.net.12.50.ssh > my.net.2.233.0: S
291769782:291769782(0) ack 1 win 5840 <mss 1460> (DF) (ttl 62, id 0)

11:10:00.247867 eth0 > my.net.2.233.0 > my.net.12.50.16: S 0:8(8) win 512
(ttl 64, id 7083)
11:10:00.248604 eth0 < my.net.12.50.16 > my.net.2.233.0: R 0:0(0) ack 9
win 0 (DF) (ttl 62, id 0)

11:22:38.393022 eth0 > my.net.2.233.0 > my.net.13.14.ssh: S 0:16(16) win
512 (ttl 64, id 13829)
11:22:38.394016 eth0 < my.net.13.14.ssh > my.net.2.233.0: S
4179305150:4179305150(0) ack 1 win 65535 <mss 1460> (DF) (ttl 126, id
30187)
11:22:38.394091 eth0 > my.net.2.233.0 > my.net.13.14.ssh: R 1:1(0) win 0
(ttl 255, id 21409)
```

As can be seen the abnormal TCP header length are not reproduced in the answer from either box. Interestingly Linux do not answer when the SYN packet has bogus header length, while Windows seems to ignore this (parameter --tcpoff 3). Both stacks sends a RST-ACK packet with sequence and acknowledgement number set to 0:0. In the Windows case the packet size is set to 16, not 8 as specified by the -d 8 parameter. I think this must because of the bogus header length.

hping2 has no option for putting bits in the TCP reserved field. To simulate this, I used another method - a poor man's packet crafter. I replaced the addresses in the capture file with addresses in my own firms network - converting them to hexadesimal 32 bits format and binary edited the capture file using hexl-mod in emacs. Then I played back the packets between the two boxes with:

```
tcpreplay-1.4.4/tcpreplay -i eth0modified_capturefile
```

Here the sender was Linux RedHat 8.0, the recipient Linux RedHat 7.0. But the packets were dropped without response just as expected.

Below is dump of the packets as sniffed from my own network:

```
tcpdump -nvXr capture-file
13:23:32.043941 my.net.12.50.16 > my.net.2.233.0: R [bad tcp cksum 1d73!]
0:8(8) ack 0 win 12450 [RST 4.0.....] (ttl 49, id 18609, len 40, bad
cksum a814!)
0x0000   4500 0028 48b1 0000 3106 a814 8b69 0c32        E..(H...1....i.2
0x0010   8b69 02e9 0010 0000 0000 0000 0000 0000        .i..............
0x0020   3414 30a2 0214 0000 0000 0000 0000             4.0...........
13:23:32.044519 my.net.12.50.16 > my.net.2.233.0: R [bad tcp cksum 1d73!]
0:8(8) ack 1 win 12450 [RST 4.0.....] (ttl 49, id 18609, len 40, bad
cksum a814!)
0x0000   4500 0028 48b1 0000 3106 a814 8b69 0c32        E..(H...1....i.2
0x0010   8b69 02e9 0010 0000 0000 0000 0000 0000        .i..............
0x0020   3414 30a2 0214 0000 0000 0000 0000             4.0...........
```

In conclusion - I found no way the detected packets could have been made by the source as normal response to an abnormal stimulus. Note also that the three addresses involved in this test are lab machines in a protected internal network.

6. Correlations

Several GCIA candidates have written about port 0 scans - I have noted Ronald Clark, and Jason Thompson. I have found no one mentioning all the abnormalities here occurring at the same time. Both note that since this port is not assigned, it will normally result in a reset, and that this might be used for recognizance.

7 Evidence of active targeting

The evidence here seems to be too inconclusive to say anything about this, and which address is meant to be the victim. Besides I have no information on the destination addresses.

8. Severity

Since I don't know what this is I cannot say anything definitive about severity. But assume that this is a test of the inverse scan I mentioned in section 3 using ICMP host unreachable to map addresses that are not used. Of course this must be based on somewhat random assumptions about the home network. So this must be seen as an example:

Criticality - I have no knowledge of the three destination addresses - no other reference to them was found in the Raw files. Then to be on the certain side we assume them to be critical - I put this at 5.

Lethality - I put this at 2 - not too lethal if not combined with a powerful attack after information gathering.

System countermeasures - assume we are ensured that all boxes are hardened, but do not have full confidence in this. I put this at 2.

Network countermeasures - assume that firewalls let out ICMP host unreachable
- I put this at 1

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)
= (5 +2) - (2+1) = 4

If a better result is desired something must be done to the countermeasures.

9. Defensive recommendation

As mentioned in the previous section the most important defense here is to avoid ICMP messages to the world. As mentioned in part 1, it is possible to block all ICMP, except Type 3 code 4 "Fragmentation needed but don't fragment bit set". Alternatively one might allow outgoing pings by using stateful inspection firewalls, and letting them control that incoming echo replies correspond to outgoing requests.

In addition those responsible for the source (the assumed router 24.206.159.155) should be notified about something fishy going on - but not a year afterwards!

The three recipients might be checked to see if they are compromised. I think however that in practice this often would be skipped in a busy network administration environment, if there is no other evidence of wrongdoing than the six packets.

10. Multiple choice test question

How can one find what has been officially assigned to a TCP or UDP well known port?

a) Read */etc/services*
b) Read OS documentation
c) Consult RFC 1700
d) Consult *http://www.iana.org/assignments/port-numbers*

Correct answer is d) On this site one can read: "Assigned Numbers: RFC 1700 is Replaced by an On-line Database"

Detect #3 - Stumbler/55808 Trojan

1. Source of the trace

For my third detect, I thought it might me interesting to try to record the activities of the Stumbler/55808 in the network of my firm. Since we have a fairly large footprint (around a million addresses) in one corner of the Internet - Norway, this might give a different picture than from other places.

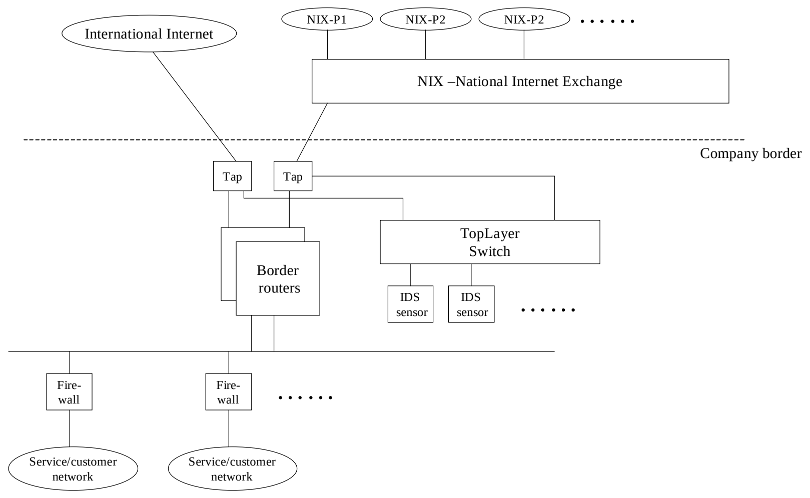

Figure 2. Conceptual graph of IDS configuration in my firm.

Now a few words on our Internet access and the IDS tap there. We get most of the Norwegian Internet traffic from something called a NIX (Norwegian Internet eXchange). This is simply an ethernet switch with connection from most Norwegian providers, as shown in the figure. It is a cooperative effort supported by the providers. Besides these, most of the larger providers have other peering connections to handle traffic from and to other countries etc. With us this is represented by the connection marked *"International Internet"*.

The IDS tap is placed on the outside of our border router. This makes it possible for me to record from which provider the packets come, just by observing the MAC address. (The placement also means a lot of noise, which we have to filter away)

One comment on naming here - this trojan have no less than three different names: Stumbler, 55808 Trojan and Typot. This is I think a sad sign of the rivalries between various security firms and research groups - it seems to be a permanent problem with almost every security problem discovered the last years. To nonexperts this must be confusing. There seems to be a need for more standardisation effort here.

Note: Another source of confusion is the wireless network sniffer Net (or Network) Stumbler, which have nothing to do with the trojan analysed here

2. Detect was generated by

The packet log and alerts was generated by a snort run for 23 hours (July 15 - 16) with the following signature, due to Guillaume in the whithats.com Snort forum:

```
alert tcp any any -> any any (msg: "Stumbler Scan"; flags: S; window:
55808; classtype: bad-unknown;)
```

The snort run also did some preprocessor logging, which I had neglected to turn off. Later I remembered that I could as well simply have used this *tcpdump* command.

```
tcpdump -nvi eth1 -w logfile tcp[14:2]=0xda00
```

The hex expression here mean: Test if the TCP window size header field offset 13 equals 0xda00, which is 55808 decimal.

I used the latter command to see if the trojan was still active on September 3, and it was. But the traffic was reduced to one fifth of what it was in June. The composition of the header was the same as before. I wonder if anybody is observing it. There is no one commenting it anymore - all comments in security forums like Bugtraq, from Intrusec etc. are from late June. If someone decided to modify it to a more dangerous form this might be a bit of a cry wolf situation.

3. Probability the address was spoofed

Here we can say with some confidence that the addresses are spoofed, since it is characteristic of the trojan that the source are spoofed. For this - see the next subsection on what is known from captured code. There are however, as we shall see false positives - normal web traffic or similar.

4. Description of the attack

Here are the count of the Stumbler detects sorted on provider. I have changed the names of the MAC-addresses of the providers to nix-p1 etc. since I haven't asked them if I could use their names.

```
  13162 nix-p1
  12706 nix-p2
    896 nix-p3
    823 nix-p4
    524 nix-p5
    491 nix-p6
    209 nix-p7
    188 nix-p8
     49 nix-p9
     22 nix-p10
     18 nix-p11
```

The distribution here probably reflects nothing but the peering routes of the providers. I am not able to go into BGP routing in this practical unfortunately, but we could possibly have learned more if we could correlate this with the routes.

Note that there is no Stumbler traffic at all via the so-called "International Internet" link we have, which is the peering link we use to non-Norwegian parts of the network. This link does not go via the NIX. I do not know why this link has no such traffic. Note also that there are a few false positives - this proved to be ordinary web traffic coincidentally having window size 55808 when I checked the access logs. There I found normal GET requests fitting addresses and timestamps, which only by a fantastic stretch of the imagination might have been made by any trojan.

It seemed a bit interesting to see how the counts of the source IP addresses were distributed. This I did with the following horrible multiple-pipe command (the percentages are added by hand). Note that I removed the destination port 80 packets here to cut down on false positives.

```
grep -v '\.http: ' stumbler_tcpd-nevr160703|xtr_ipad|cut -d " " -f
1|sort|uniq -c|sort -nr|cut -f 1|sort -n|uniq -c
9563 1 33%
4493 2 15.5 %
1939 3 6.7 %
 627 4 2.2 %
 127 5 0.4%
  62 6 0.2 %
   7 7 -
   2 9 -
   1 11 -
   1 12 -
etc....
```

Now I get a uniq-sort on the frequencies of the frequencies (This means that 9563 addresses occurs one time etc.) As can be seen the addresses seem to be quite uniformly distributed, but a stringent statistical analysis of this is outside the scope of this detect I think. The least frequent tail in the listing here might represent more false positives, as I have only grep'ed away the http requests.

Of course the destination address frequencies here would be quite differently distributed, since they have been sorted by the routing of the network.

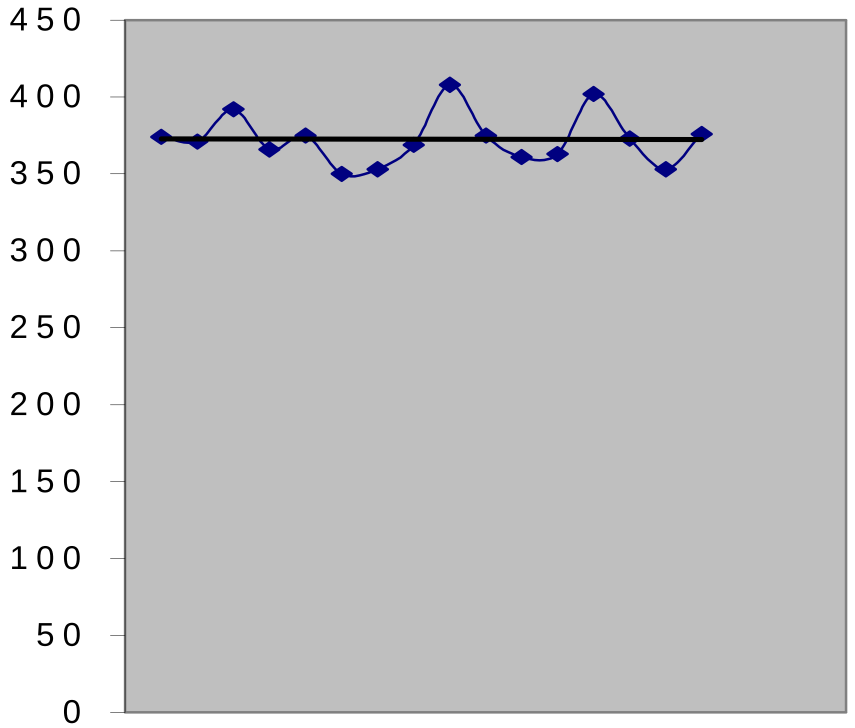

Figure 3. Graph of how the destination addresses are distributed.

Figure 3 above is meant to show the distribution of the destination addresses. This I made by simply truncating the addresses before the third octet, and making a count over 16 B-networks. As can be seen the distribution is fairly even, as given by the trend line, but a larger sample should be made to decide this more certainly. (And even better - statistical testing methods could be used, but that is far beyond my competence)

```
[root@minestrone kbjo]# p0f-current/p0f  -s stumbler.log2|cut -d " "
-f 3-|sort |uniq -c|sort -nr
.....
   8949 [55808:122:1460:0:2:1:1:52].
   8769 [55808:121:1460:0:2:1:1:52].
   6429 [55808:120:1460:0:2:1:1:52].
   1834 [55808:123:1460:0:2:1:1:52].
    898 [55808:126:1460:0:2:1:1:52].
    824 [55808:125:1460:0:2:1:1:52].
    794 [55808:119:1460:0:2:1:1:52].
    205 [55808:124:1460:0:2:1:1:52].
    183 [55808:118:1460:0:2:1:1:52].
    156 [55808:114:1460:0:2:1:1:52].
     49 [55808:125:1460:1:2:1:1:52].
     19 [55808:122:1460:1:2:1:1:52].
      7 [55808:119:1372:1:2:1:1:52].
      6 [55808:120:1452:1:2:1:1:52].
      5 [55808:123:1460:1:2:1:1:52].
      5 [55808:115:1460:0:2:1:1:52].
      3 [55808:123:1402:0:2:1:1:52].
      3 [55808:119:1452:1:2:1:1:52].
      2 [55808:121:1460:1:2:1:1:52].
      1 [55808:123:1402:0:0:1:1:52].

The parameters here are wwww:ttl:mmm:D:W:S:N:I:OS
where
# wwww - window size
# ttl  - time to live
# mmm  - maximum segment size
# D    - don't fragment flag  (0=unset, 1=set)
# W    - window scaling (-1=not present, other=value)
# S    - sackOK flag (0=unset, 1=set)
# N    - nop flag (0=unset, 1=set)
# I    - packet size (-1 = irrevelant)
```

Above is a p0f run on the data.

Now I think this might be a bit interesting. The only two parameters who differs here is the TTL and the DF flag, and this might suggest a common platform, or maybe that they all are from the same variant. Of course - since these are crafted packets, this might be misleading too. This should be tested against the captured variant, but I have not found any reference to something like this having been investigated. The code captured by Intrusec ran on Linux (they say nothing on which kernel). There are 34 Linux varieties in the p0f profile base, most have ttl 64, mmm1460, D 1, S 1, N 1. W varies but few have 2 and I varies but none are 52. So this doesn't fit a Linux platform too well. Note also that some of the difference her might be due to false positives. Note that according to [50], window scaling is set to 2 by some variant - the value that is here in all but one packet.

Here is the distribution of TTL's in the run:

```
tcpdump -nvr stumbler.log2|grep -v '\.http: '|perl -ne
'/\(ttl (\d+),/;print "$1\n"' |sort|uniq -c|sort -nr
   8960 122
   8769 121
   6429 120
   1838 123
    898 126
    824 125
    794 119
    205 124
    183 118
    156 114
      5 115
```

If one assumes the original TTL of the packets are 128 (but it could be anything, being crafted actually), then 898 of them are just two hops from our network, 824 are 3 hops away. I have listed a count of these packets:

```
[root@minestrone kbjo]# tcpdump -nevr stumbler.log2|egrep
'ttl 126,|ttl 125,'|bin/maxip_xtr |awk '{print
$1}'|sort|uniq -c|sort -nr
    898 nix-p3
    824 nix-p4
     49 nix-p10
```

Now it might be possible to contact these three providers, and cooperate on catching the trojan. But I haven't tried this, getting in contact with them and agreeing on this is a major undertaking, I think.

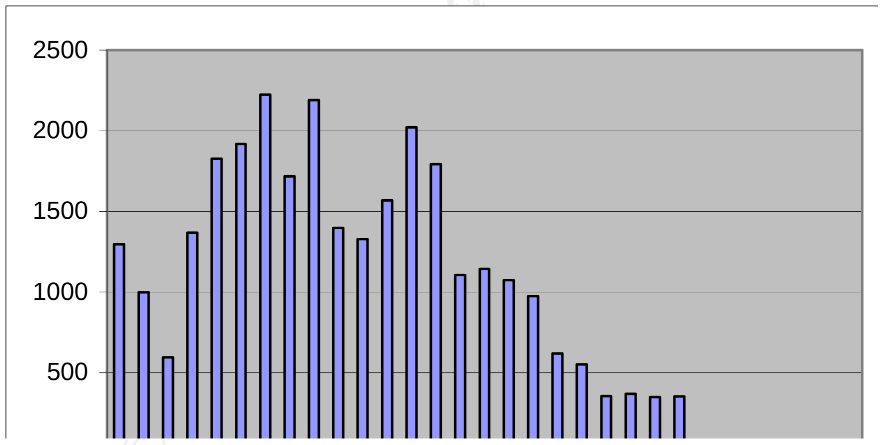

Figure 4. Chart showing packet distribution.

Above is a chart showing packet distribution pr hour starting at 7:00 Middle European time (UTC +2) the first day and ending 6:00 the next day. This has been made simply with a *sort-uniq -c* command on the hour field from tcpdump and cut and pasted into MS Excel.

Strangely there is more traffic at daytime than at night, but this might conceivably reflect the traffic condition where the trojan is installed. Then one might think that it might be placed somewhere where business hours are 7 hours before ours - somewhere in Asia perhaps?

5 Attack mechanism

A variant of Stumbler was captured by Intrusec and analysed by them (see http://www.intrusec.com/55808.html) They say it looks like proof of concept code, doing nothing harmful, but scanning the network ineffectively by sending and receiving packets with window size 55808 and with spoofed IP addresses. Or rather - both source and destination are random. It had code for transmitting the scan results to a specified address over port 22, but this wasn't functioning.

CERT [50] mentions three variants, the one captured by Intrusec, the one captured by ISS, and a variant of the IRC bot Trojan sdbot. They say that "not all of them" set the winscale option to 2, but give no further detail.

It has also code for getting a new receiving IP-address for the data it collects, coded into the sequence number but this seemed not to function in the captured Linux binary.
It has no code for attacking the hosts it resides on, so it must be put in position by some external automated or manual attack.

Since the source is spoofed, the answering ACK's and RST's will not go back to the sending trojan, but must be picked up by another trojan. Unless there is a really staggering infection rate only a very tiny part of the responses will be captured since there has to be another trojan in the return path, sniffing the interface on the address it is installed. Theoretically there is around 3.7 billion addresses on the Internet. Since we get around 1300 packets pr. hour, and has less than one millionth of the entire Internet, the trojans must generate several billion packages pr. hour. This supposes that the traffic is evenly distributed over the entire Internet.

But an individual trojan will not even see one package every hour this way, so scanning all addresses might take ages, depending somewhat on where they are installed. If some of them are on border gateways, or some other receiving mechanism are installed this way, they will be much more efficient. Actually there is scant proof that most of these trojans really are on compromised machines. They may as well be on machines owned by those spreading them in some cases.

6. Correlations

This doesn't seem to have been commented on by any GCIA candidates, but there was some discussion in various security forums in late June e.g. Bugtraq . Several security organizations, firms and news media reported it. Later, as the trojan didn't seem to be doing much, interest has diminished.Reference [32 -37] describes this trojan.

7. Evidence of active targeting

The trojan in its known variant have very little targeting at all. It does not even have a way to transfer the scanning data it collects, because of the malfunctioning of the upload IP address transmitting.

Some people have suggested that it must be some kind of prank against the security community. Alternatively it might also be an attempt to measure the reaction to a harmless variant, to check how prepared the security community are to handle advanced distributed malware. At least I find it difficult to believe that anyone would think seriously to scan the Internet in such a stupid way.

Of course the machines where the trojan is running might have been compromised, but even this is not sure, as mentioned.

8. Severity

The calculation her should be straightforward, but a bit pointless:
Criticality - since all machines are targeted this must be 5
Lethality - since the traffic seems harmless this might be 1, but i put it to 2 to account for the cry wolf effect.
System countermeasures - none are needed - 5
Network countermeasures - none are needed - except a watch to observe if the packets change content. To account for the possibility that we miss a dangerous change I put this at 4.
(5+1)-(5+4)=-3

9. Defensive recommendation

This and similar technology using a net of distributed trojans, communicating with crafted packets that break the rules of ordinary traffic to be stealthy might be a growing problem. Other toolkits within the same family are the DDOS tools, TFN, Trinoo etc. and the trojan Q, analysed by Les Gordon in his practical.

One can also say that the worm Nimda belongs to this group, even if its networking capacity was not used - just like the captured Stumbler variant. A doomsday scenario is that a permanent underground network are set up using tools like this, and perhaps using a worm as infection vector. Just think about the persistence of the old Code Red and other worms we see today - this might give an idea about how difficult this would be to fight. This network might then be used as a platform for all kinds of attacks, not just DDOS attacks. One idea is that an attacker newer need to use real IP addresses, since the trojans will pick up everything.

A possible positive factor is that to this day nobody has come up with a tool like this that works really well, as far as is known. To get a better score against distributed trojan tools I think a better cooperation between the operators and providers is needed. Today this is very weak. In my analysis here I hope I have come up with some ideas that could be built on. As long as an investigation like this has to stop at the organization border, little will be achieved.

Maybe one could agree on deploying distributed IDS tools that can collect information from so many points in the Internet that it might be covered. But to achieve this, one would have to break down lots of corporate and national barriers I should think

10. Multiple choice test question

What is the function of the window size field in TCP
a) Used for sizing GUI windows
b) Used for timing the packets
c) Giving the size of the datagram.
d) The number of bytes a receiver is willing to accept, providing flow control.

d) Should be the right answer according to W. Richard Stevens

Analyse this

Overview

I have analysed the logs from the NIDS sensor of the University, where I comment on the details in the alert, port scan and out of spec files. At the end I conclude that the University should take several steps to tighten security.

Among the things I recommend are inspecting several servers for compromise, revising firewall and routing policies, and setting up a better rule set for the NIDS sensor. Besides I suggest better cooperation within system and network administration on security matters.

This analysis is based on a relatively limited knowledge of the University, its network and security work. Also I have little knowledge of security at universities in general, and US universities in particular, having a background in business security in Europe. But I understand that there might be a pressure towards a generally more liberal security policy than what is common within business. This must be weighed against the increasing problem with massive attacks from the Internet, sometimes leading to total denial of service, like we have seen during recent worm attacks.

Analysed files

The following files from July 2 to July 7, 2003 were downloaded for analysis. There was no alert file for July 4, and no OOS file for July the 2

```
alert.030702.gz
scans.030702.gz
OOS_Report_2003_07_04_14486
alert.030703.gz
scans.030703.gz
OOS_Report_2003_07_05_3053
scans.030704.gz
OOS_Report_2003_07_06_23454
alert.030705.gz
scans.030705.gz
OOS_Report_2003_07_07_25549
alert.030706.gz
scans.030706.gz
OOS_Report_2003_07_08_5584
alert.030707.gz
scans.030707.gz
```

Alert frequencies

Observe that the alert messages must come from a highly customized rule set . Since I don't know the snort rules used here (no SIDS!), I must guess what they are from the messages. But in some cases I compare them with the default rules never the less.

There are some broken log lines in the alert files - leading to inconsistencies like the 253 empty messages below. They had only part of the IP:PORT fields at the end - like this

```
:137
:27377 -> MY.NET.100.165:80
:36869 -> MY.NET.100.165:80
:3403 -> MY.NET.100.165:21
:80
.......

[kbjo@minestrone logs]$ grep '^:' alert.03070?|wc -l
    278
```

I decided to ignore them by greping them away. There was also some records that was truncated and lacked LF leading to corruption of the following records. These I corrected by removing the corrupt part. The corrected alert records was put in a file *alert5.corr2*

Top ten talking pairs

```
67065 205.160.101.121 MY.NET.83.100
 9532 MY.NET.153.185 218.153.6.197
 7423 MY.NET.153.185 218.153.6.212
 4199 65.214.36.116 MY.NET.100.165
 3274 MY.NET.97.188 202.103.69.100
 3112 MY.NET.97.38 65.127.129.10
 3072 MY.NET.83.100 64.235.110.34
 2830 MY.NET.97.60 202.103.69.100
 2608 MY.NET.83.100 208.194.163.37
 2330 MY.NET.111.34 63.164.243.132
```

Top ten alert sources	Top ten alert destinations
67070 205.160.101.121	149378 MY.NET.100.165
18447 MY.NET.153.185	67088 MY.NET.83.100
8198 MY.NET.83.100	17037 MY.NET.30.4
5440 169.254.45.176	9539 218.153.6.197
4626 65.214.36.116	8757 202.103.69.100
3853 MY.NET.97.188	7426 218.153.6.212
3170 MY.NET.97.38	4909 MY.NET.137.7
2953 MY.NET.97.60	3112 65.127.129.10
2421 209.172.113.153	3072 64.235.110.34
2353 MY.NET.111.34	2911 MY.NET.190.93

Table 4. Top ten totals after corrupt records and spp_portscan alerts has been removed

Relations between the addresses

The possibilities for analysing the relations between machines are limited to what are in the files. The alert files have to be the main source, port scans and probes don't tell too much about this (But see below about MY.NET.1.3 and 4) Besides, the machine with most alerts are not always the most important in the network, some busy servers can be totally silent in this respect. On the other hand a probe on a certain port e.g. 25 does not mean that this service are present and that this is a mail server. But here is what I found showing the 5 most frequent of the following categories, and assuming that the alerts mostly reflect what the addresses really are.

Most alerted MY.NET Web servers	Most alerted MY.NET FTP servers

```
  40966 MY.NET.100.165
  15460 MY.NET.30.4
   1148 MY.NET.86.19
    700 MY.NET.24.44
    683 MY.NET.110.224
```
	```
	57 194.66.54.193   MY.NET.100.165
	 7 62.101.125.237  MY.NET.100.165
	 7 213.140.20.187  MY.NET.100.165
	 6 218.166.204.203 MY.NET.100.165
	```

Most alerted outgoing Web accesses	**Most alerted outside Web servers**

```
  18447 MY.NET.153.185
   3853 MY.NET.97.188
   3170 MY.NET.97.38
   2953 MY.NET.97.60
   1619 MY.NET.98.99
```
	```
	   9538 218.153.6.197
	   8756 202.103.69.100
	   7426 218.153.6.212
	   3112 65.127.129.10
	   1991 207.200.86.66
	```

MY.NET.1.3, MY.NET.1.4 and *MY.NET.1.5* seem to be DNS servers if their occurrence in the scan files is a typical port 53 false positive. *MY.NET.100.165* - the CS Web Server should be no surprise here, since all access is alerted. But since I don't know the reason for this, it is hard to comment on this. *MY.NET.69.145* might be an important mail server, since there are several alerts concerning port 25. *MY.NET.24.47* and *MY.NET.99.5* (row 33 and 59 in the alert table 6 below) seems to be FTP servers. *MY.NET.3.54* and *MY.NET.3.56* (row 38 and 42) must have Web servers. *MY.NET.7.49* and *MY.NET.7.50* are serving the Help Desk, and have FTP servers.

Internet Relay Chat traffic

The following alerts show that IRC is much used in the University network:

```
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC
[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC
bot
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.
[UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan.
[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC
bot
IRC evil - running XDCC
```

The existence of all these rules seems to indicate that IRC are used for serious purposes at the Univerity. The rules give the impression of attempting to control the use of IRC.

Unfortunately I know very little of this protocol, having barely touched an IRC client! In my home network IRC is mostly outlawed. I think this is typical of a business environment. But of course the protocol might be used for quite serious purposes, even if it has a reputation of being a paradise for teenage script kiddies and the like. The treatment here is based on reading about the protocol in connection with this practical and must necessarily be a bit theoretical and superficial.

There seem to be problems with the way IRC are used. In the "Conclusions" subsection I suggest a few ideas of what can be done about this, but someone with more knowledge of IRC must be consulted for further advice on this.

Peculiar to IRC is the way the person or persons who control an IRC channel - a channel operator (op or oper) are decided. This role must not be confused with the administrator of an IRC server who also is called operator, op or oper. Usually there is an algorithm to decide who is channel op, and users or idents that are not present in the channel for a time are excluded from this position. This makes it possible to make a kind of game or warfare out of it. Illegal intrusions on other computers are done to get weapons against other channel op's.

So-called IRC bots can do this. An IRC bot is a client simulator, which might be controlled over the Internet. They are used for several purposes, not all of them illegal. But a number of them might be installed (usually illegally) around the Internet and make a DOS attack, usually by so-called flooding. Large number of characters is sent to the channel making the server throwing out the ident that the bots spoof. Alternatively more conventional DOS attacks on the address of a client or server might be done or trojans might be installed on a client address to capture passwords and attack other users. Actually I understand that much of the motive for cracking activity is IRC warfare, which might explain that one often finds IRC bots on compromised computers.

Another related activity is file sharing over IRC . Some of this is pirated software, music and video files - often called "warez". The IRC file transfer protocol DCC uses bots. When the files transferred are illegal this is often done via bots on compromised computers.

Here is the top ten IRC alert generators:

```
  33443 205.160.101.121 MY.NET.83.100
   3072 MY.NET.83.100 64.235.110.34
   2608 MY.NET.83.100 208.194.163.37
   1684 MY.NET.83.100 205.160.101.121
   1617 MY.NET.84.228 206.167.75.78
   1607 206.167.75.78 MY.NET.84.228
    833 MY.NET.83.100 155.207.19.204
    489 160.94.151.137 MY.NET.150.218
    483 194.159.164.195 MY.NET.150.218
    467 64.62.96.34 MY.NET.150.218
```

I made the following link graph between the eight most frequent addresses:

Linking adresses with more than 300 IRC alerts

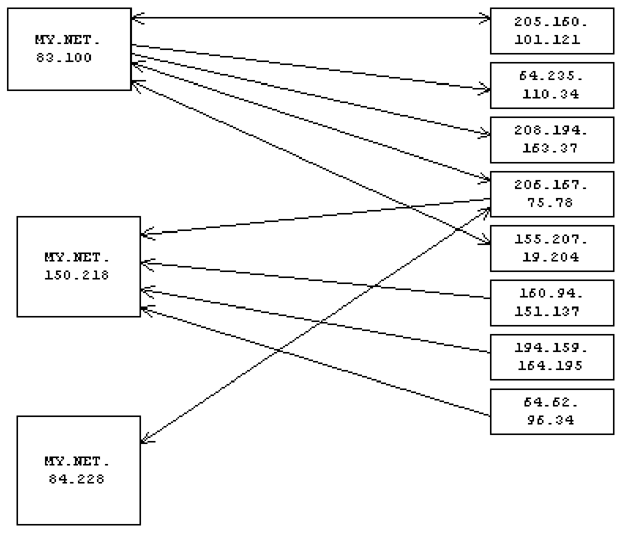

Figure 5. Link graph.

Information on the eight addresses - some of the names suggest designated IRC servers:

Adress	DNS hostname	Whois information
205.160.101.121	irc.rma.edu	Randolph Macon Academy
64.235.110.34	saturn.izolnetworks.com	Packetworks Inc. 1
208.194.163.37	twisted.irctoo.net	First Internet Alliance
206.167.75.78	cricri.qeast.net	Reseau Interordinateur Scientifique Quebecois1, Canada
155.207.19.204	egnatia4.ee.auth.gr	Aristotle University of Thessaloniki, Greece
160.94.151.137	babblex.tc.umn.edu	University of Minnesota
194.159.164.195	efnet.demon.co.uk	Demon Internet Limited, UK
64.62.96.34		Axient Communications, Inc.

Table 5. IRC addresses

Frequencies in the five alert files

Row	# of alerts	Message	3 most frequent address pair	DST port
1	46111	spp_portscan	The number reflects the "PORTSCAN DETECTED" alert. "End of portscan" is slightly lower at 44593. See subsection on scanning	-

2	71184	spp_http_decode: IIS Unicode attack detected	See comment below	80
3	3582	spp_http_decode: CGI Null Byte attack detected	236 MY.NET.97.139 165.193.133.57 215 MY.NET.97.238 217.174.99.5 200 MY.NET.141.225 216.15.92.7	80, 8080
4	140537	CS WEBSERVER - external web traffic	4199 65.214.36.116 MY.NET.100.165 1345 216.88.158.142 MY.NET.100.165 1221 66.147.154.3 MY.NET.100.165	80
5	64406	SMB Name Wildcard	1942 213.65.76.114 MY.NET.137.7 1622 213.204.59.157 MY.NET.137.7 865 24.117.55.43 MY.NET.137.7	137
6	44268	Queso fingerprint	See comment below	113, 4667, 80,25
7	40380	[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	33443 205.160.101.121 MY.NET.83.100 1607 206.167.75.78 MY.NET.84.228 489 160.94.151.137 MY.NET.150.218	vari-ous
8	17037	MY.NET.30.4 activity	739 172.172.54.149 MY.NET.30.4 427 65.214.36.116 MY.NET.30.4 334 66.196.65.37 MY.NET.30.4	vari-ous
9	9099	EXPLOIT x86 NOOP	1615 64.122.109.8 MY.NET.190.93 988 130.83.206.1 MY.NET.190.93 536 168.26.240.26 MY.NET.110.224	80, 139
10	8403	CS WEBSERVER - external ftp traffic	670 213.140.8.172 MY.NET.100.165 630 213.140.15.170 MY.NET.100.165 487 213.140.12.216 MY.NET.100.165	21
11	8202	[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	3072 MY.NET.83.100 64.235.110.34 2608 MY.NET.83.100 208.194.163.37 1684 MY.NET.83.100 205.160.101.121	6667
12	5246	High port 65535 tcp - possible Red Worm - traffic	2330 MY.NET.111.34 63.164.243.132 1576 63.164.243.132 MY.NET.111.34 149 MY.NET.97.93 217.209.142.239	65535
13	2752	MY.NET.30.3 activity	779 68.55.226.150 MY.NET.30.3 557 68.49.35.0 MY.NET.30.3 455 68.55.52.234 MY.NET.30.3	vari-ous
14	1864	[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	1617 MY.NET.84.228 206.167.75.78 146 MY.NET.153.113 129.143.67.242 82 MY.NET.153.113 206.252.192.195	6667
15	1491	connect to 515 from inside	1491 MY.NET.162.41 128.183.110.242	515
16	1416	External RPC call	See comment below	111
17	1373	TCP SRC and DST outside network	See comment below	vari-ous
18	1361	IDS552/web-iis_IIS ISAPI Overflow ida nosize	15 218.5.66.254 MY.NET.115.28 8 200.198.136.53 MY.NET.24.35 7 130.13.151.35 MY.NET.69.192	80
19	937	Null scan!	133 213.176.8.2 MY.NET.25.73 112 63.251.52.75 MY.NET.150.203 106 67.119.233.217 MY.NET.12.4	0,110
20	850	IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	7 MY.NET.69.145 130.126.118.103 6 MY.NET.97.61 130.223.88.237 6 MY.NET.97.61 130.127.77.214	80
21	781	NMAP TCP ping!	83 193.41.181.254 MY.NET.112.195 72 63.211.17.228 MY.NET.1.3 68 64.152.70.68 MY.NET.1.3	

22	709	High port 65535 udp - possible Red Worm - traffic	59 MY.NET.153.223 218.127.161.81 56 63.250.195.10 MY.NET.150.203 50 MY.NET.150.242 80.15.155.6	65535
23	585	connect to 515 from outside	See comment to row 15	525
24	487	NIMDA - Attempt to execute cmd from campus host	See comment below	80
25	392	SUNRPC highport access!	108 68.50.111.222 MY.NET.101.44 74 66.7.161.3 MY.NET.97.92 40 194.109.217.138 MY.NET.97.216	32771
26	376	Possible trojan server activity	20 MY.NET.24.44 213.190.192.120 20 129.41.69.86 MY.NET.25.70 17 MY.NET.12.4 68.32.63.62	27374
27	346	SMB C access	3 81.212.30.214 MY.NET.132.45 3 68.117.146.57 MY.NET.132.45 3 67.35.116.205 MY.NET.152.252	139
28	264	EXPLOIT x86 stealth noop	229 129.165.254.6 MY.NET.163.143 12 131.118.254.130 MY.NET.24.8 11 129.49.105.43 MY.NET.60.11	vari-ous
29	208	Incomplete Packet Fragments Discarded	29 151.196.121.201 MY.NET.11.4 25 151.196.177.2 MY.NET.11.4 24 MY.NET.83.98 151.196.19.26	0
30	187	SNMP public access	SNMP access should bee blocked from outside. Only one address pair: 134.192.86.65 MY.NET.190.13	161
31	136	NIMDA - Attempt to execute root from campus host	See comment below	80
32	134	TFTP - Internal TCP connection to external tftp server	See comment below	69
33	70	FTP passwd attempt	21 217.228.31.172 MY.NET.24.47 9 199.243.85.90 MY.NET.24.47 7 12.47.47.2 MY.NET.24.47	21
34	59	TFTP - Internal UDP connection to external tftp server	8 MY.NET.83.69 216.17.103.14 8 MY.NET.5.92 81.171.2.192 8 63.250.195.10 MY.NET.150.203	69
35	57	EXPLOIT x86 setuid 0	7 63.240.202.73 MY.NET.97.187 3 140.254.73.38 MY.NET.84.22 2 81.101.247.56 MY.NET.111.51	vari-ous
36	44	EXPLOIT NTPDX buffer overflow	28 63.250.195.10 MY.NET.150.203 4 198.64.140.205 MY.NET.97.44 3 206.204.200.108 MY.NET.18.22	123
37	43	Tiny Fragments - Possible Hostile Activity	29 61.171.249.46 MY.NET.25.70 7 24.242.101.56 MY.NET.99.48 5 MY.NET.114.120 217.224.247.198	no ports
38	39	Notify Brian B. 3.54 tcp	Count seems without interest	80
39	39	ICMP SRC and DST outside network	See comment below	no ports
40	35	IRC evil - running XDCC	See comment below	6667
41	34	EXPLOIT x86 setgid 0	3 131.118.254.130 MY.NET.24.8 2 66.119.34.38 MY.NET.97.179 2 216.168.224.69 MY.NET.153.210	80,119
42	31	Notify Brian B. 3.56 tcp	Count seems without interest	

43	19	NETBIOS NT NULL session	2 210.96.88.129 MY.NET.137.46 2 210.96.88.129 MY.NET.137.37 2 210.96.88.129 MY.NET.137.36	139
44	17	RFB - Possible WinVNC - 010708-1	See comment below	5900, 5901
45	14	DDOS shaft client to handler	4 211.43.197.9 MY.NET.110.80 4 207.69.200.159 MY.NET.6.55 4 132.174.11.11 MY.NET.24.27	25,80
46	14	Attempted Sun RPC high port access	4 63.250.195.10 MY.NET.150.203 2 63.250.195.10 MY.NET.117.10 2 208.172.128.163 MY.NET.74.247	32771
47	11	Traffic from port 53 to port 123	11 64.125.197.7 MY.NET.1.3	53,123
48	11	SYN-FIN scan!	8 63.251.52.75 MY.NET.150.203 1 80.11.69.141 MY.NET.112.180 1 63.251.52.75 MY.NET.97.33 1 63.251.52.75 MY.NET.87.131	vari-ous
49	11	External FTP to HelpDesk MY.NET.70.50	See comment below	21
50	10	[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot	See comment below	6667
51	10	External FTP to HelpDesk MY.NET.70.49	See comment below	21
52	9	TCP SMTP Source Port traffic	See comment below	
53	8	[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	See comment below	6667
54	4	[UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan.	See comment below	6667
55	3	[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	See comment below	6667
56	3	Probable NMAP fingerprint attempt	See comment below	vari-ous
57	2	DDOS mstream handler to client	Se comment on this in Bjornstad Practical part 1	15104
58	1	TFTP - External UDP connection to internal tftp server	See comment below	69
59	1	FTP .forward	Server must be checked - no FTP server should allow this from a user home directory	21
60	1	Fragmentation Overflow Attack	Don't know enough to comment - one alert seems too little to investigate	
61	1	External FTP to HelpDesk	Don't know enough to comment	

		MY.NET.53.29		
62	1	EXPLOIT x86 NOPS	See comment 5 below	
63	1	CS WEBSERVER - external ssh traffic	Don't know enough to comment	
64	1	Back Orifice	This should of course be investigated	

Table 6. Count of the alert in the University's alert files. More detail of

Comments to the table

Row 2: spp_http_decode: IIS Unicode attack detected
This might be a sign of Nimda, which uses a range of Unicode's (as Microsoft interprets them), exploiting at least two vulnerabilities in unpatched Windows boxes. Here it is the preprocessor http_decode that gives the alert and this is a bit hard to analyse. Preprosessor alerts are very sparsely documented, and I found no precise description of this alert.

Les Gordon [40] commented on this mentioning other comments by Todd A.Beardsley [38], Colin Carpenter, Christopher Lee, and Bradley Urwiller. Gordon notes that false positives might occur in connection with Asian sites (and in other parts of the world with websites targeted at Asians) and when using SSL.

To test on the Asian site false positive theory I ran the destination addresses trough whois and grep'ed on OrgName. Of 864 unique addresses 463 was from Asia Pacific Network Information Centre, so this might very well be the case. See *http://www.apnic.net*

As Gordon notes, an ordinary signature might discover Nimda. If a signature like Snort's SID1945 is used, with content *"/..%255c.."* there should be a low incidence of false positives. The pre-processor alert might then be disabled with the *-unicode* parameter. If you do this no normalization is done, so then you must have signatures for all relevant Unicode variants - a very high number I should think.

To test the behaviour of Snort as regards Nimda signatures I ran a full list of Nimda signatures through a Snort-2.0.0 where all pre-processors except http_decode was uncommented. The signatures were lifted from an Apache access log, where we mostly ignore them. I ran the signatures from one test machine to another by using hping2 the file *~kbjo/nimdareqs* contains the GET arguments used by Nimda.

```
The signatures:
GET /scripts/root.exe?/c+dir
GET /MSADC/root.exe?/c+dir
GET /c/winnt/system32/cmd.exe?/c+dir
GET /d/winnt/system32/cmd.exe?/c+dir
GET /scripts/..%255c../winnt/system32/cmd.exe?/c+dir
GET
/_vti_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir
GET
/_mem_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir
GET
/msadc/..%255c../..%255c../..%255c/..%c1%1c../..%c1%1c../..%c1%1c../wi
nnt/system32/cmd.exe?/c+dir
GET /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%c0%2f../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%%35%63../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%%35c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%25%35%63../winnt/system32/cmd.exe?/c

hping2 statement:
for i in `cat ~kbjo/nimdareqs`; do echo 'GET' $i>/tmp/req;hping2 -c 1
-p 80 -d 31 -E /tmp/req  minestrone; done
```

The result somewhat disappointingly was only ordinary rule signatures SIDS 970, 981, 982, 983, 1002, 1242, 1286, 1945. There was no http_decode alerts. Possibly this has changed since earlier Snort versions.

To conclude: Here several MY.NET-boxes seem to be infected and should be checked on immediately. Some of he frequencies here however seem a bit too low for a worm attack - the boxes should be checked for a manual compromise too. And as mentioned above an Asian language false positive theory must be checked. This ought to be easily checked against the packet log.

I must also mention that if *MY.NET.153.185* is a web proxy this reflects infections on the inside.

```
grep 'spp_http_decode: IIS Unicode' alert.03070*|xtr_ipad|cut -d " "
-f 1|sort|uniq -c|sort -nr |head -20
  18447 MY.NET.153.185
   3853 MY.NET.97.188
   3160 MY.NET.97.38
   2953 MY.NET.97.60
   1619 MY.NET.98.99
   1315 MY.NET.97.29
   1136 MY.NET.97.243
   1116 MY.NET.97.85
   1083 MY.NET.75.107
   1015 MY.NET.152.179
    967 MY.NET.69.249
    876 MY.NET.97.34
    806 MY.NET.97.84
    793 MY.NET.97.97
    651 MY.NET.153.157
    596 MY.NET.153.114
    577 MY.NET.116.84
    564 MY.NET.97.201
    557 MY.NET.97.154
    550 MY.NET.97.13
```
Table 7. Affected MY.NET source addresses:

Row 3: spp_http_decode: CGI Null Byte attack detected
Again it is hard to find out what http_decode really alerts on here. I miss the packet logs. Several sources mention Null Byte attacks, but the only one mentioning this as CGI-specific is: [4] http://www.securityfocus.com/bid/977:
" Zeus Web Server Null Terminated Strings Vulnerability

Appending "%00" to the end of a CGI script filename will permit a remote client to view full contents of the script if the CGI module option "allow CGIs anywhere" is enabled. Scripts located in directories which are designated as executable (eg. \cgi-bin) are not vulnerable to this exploit."

From the Snort FAQ [2]:
" 4.17 I am getting too many "IIS Unicode attack detected" and/or "CGI Null Byte attack detected" false positives. How can I turn this detection off? These messages are produced by the http_decode preprocessor. If you wish to turn these checks off, add -unicode or -cginull to your http_decode preprocessor line respectively.
preprocessor http_decode: 80 8080 -unicode -cginull
Your own internal users normal surfing can trigger these alerts in the preprocessor. Netscape in particular has been known to trigger them. Instead of disabling them ,try a BPF filter to ignore your outbound http traffic such as:
snort -d -A fast -c snort.conf not (src net xxx.xxx and dst port 80)"

It is somewhat striking that many more IDS forum postings are of the kind "how could I turn off this or that alert", than about how to alert on something.

More investigation is needed here to determine if these alerts are false positives, or due to some attack. If some of the servers are using Zeus, this might be somewhat more interesting. Note also that one of the suspected Nimda victims *MY.NET.97.23* gave this alert.

Row 4 CS WEBSERVER - external web traffic
I don't know what the CS Webserver is, so I don't comment much on this. Number two on the frequency list - 142.158.88.216 (crawlers.looksmart.com) - is obviously the robot of a search engine.

Row 5: SMB Name Wildcard
According to L.Gordon and T.Beardsly these are false positives from ordinary NetBios sessions.

Row 6: *Queso fingerprint*
These must be false positives. I see no reason that Queso probes should generate all this traffic. Even if the signature also catches Nmap fingerprinting (Nmap should be much more frequent), this cannot explain the numbers. Here are the 5 most frequent addresses, with destination ports:

```
  33621 205.160.101.121 MY.NET.83.100 113
   1061 168.226.117.32 MY.NET.112.196 4662
    311 80.143.95.179 MY.NET.112.196 4662
    302 200.67.29.153 MY.NET.60.11 80
    300 200.67.29.153 MY.NET.60.38 80
```

To send 33621 packets to port 113 - auth (or ident) to fingerprint one host seems a bit excessive! Seemingly this is a false positive connected to IRC traffic, here is a count of the alerts on the two most frequent IRC talkers - *205.160.101.121 <-> MY.NET.83.100*:

```
  33624  Queso fingerprint
  33444  [UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.
   1685  [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC
```

Note that IRC usually require auth queries for each session.

There are also alerts on ports 4662 and 80. The file sharing system eDonkey (or p-to-p - similar to Kazaa) uses port 4662. File sharing might be considered a problem both because of the possibility of spreading copyrighted material through University machines, and because of security problems with the protocols - usually they are effective spreaders of worms etc.

Reference: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids29&view=research
This says false positives are a problem, but that a high TTL threshold of 225 will fix this. Probably this has not been done to the University's rule.
Teri Bidwell [41] has also mentioned Queso false positives in his practical, but in connection with port 994 - IRC over SSL.
It is not easy to guess which signature is used here, the standard Snort rule set has no Queso fingerprint rule. I has a rule for Nmap fingerprint alerting on the SFPU TCP flag combination. There are however no corresponding number of SFPU alerts in the OOS files (se following subsection). Nmap builds its method of fingerprinting on Queso, so the signature should be similar. However Nmap uses several different warped packets, so correct alerting on it is not easy. See [45], and see also the comment below on OOS alert on *12****S**

Row 7: [UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.

This is an attempt to alert on attempts to throw other users out of the IRC servers. The kill command is usually used by server op's to throw out difficult users. If this really is a trojan, it might be a systematic attempt at a DOS attack against the IRC traffic. The server op's just authenticate with an ordinary password, so their ident is not well protected. Note that there does not have to be a trojan attack here, the password can be obtained otherwise.

But it seems strange to have this many attacks of this kind. Besides it is remarkable that all the alerts have port 6667 or similar as source port with various 4-digit ports as destination. This must mean that there are alerts on IRC return packets. The kill attack packets should go the other way. Most probably this is a false positive on other traffic, for instance DCC file share traffic with a file name with full directory path where "/kill" is part of the string. This should be investigated by looking at the packet logs.

```
202.91.34.9 MY.NET.97.231
205.177.13.100 MY.NET.108.48
205.177.13.100
MY.NET.152.159
4.62.103.167 MY.NET.97.143
63.102.226.240 MY.NET.97.150
63.102.226.240 MY.NET.97.192
63.102.226.240 MY.NET.97.235
63.102.226.240 MY.NET.97.62
63.102.226.240 MY.NET.97.71
63.102.226.240 MY.NET.97.85
64.55.29.205 MY.NET.97.211
```

Note that a few of these alerts have the port 7000 that are used by a number of trojans according to www.simovits.com [28], among them the popular SubSeven. Then these might be true trojan traffic controlling bots on external addresses from the University. On the left are the talking pairs - the owners of the external addresses should be notified, and the internal addresses examined.

Row 8: MY.NET.30.4 activity

```
 15460 80
   810 524
   739 51443
    14 17300
     4 666
     4 32559
     2 443
     2 22
     1 34287
     1 21
```

I don't know the reason for this alert so I don't comment much on this. There are various destination ports here (see the listing on the left) - but except for 666 which are used by several trojans I find no negative references to the others:

Row 9: EXPLOIT x86 NOOP.

There are lots of exploits of this type, targeting various OS's on Intel boxes. It is hard to comment on this without knowing the rule. Snort 2.0 default rule set has similar rules with message SHELLCODE x86:

```
alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE
x86 NOOP"; content: "|90 90 90 90 90 90 90 90 90 90 90 90 90 90|";
depth: 128; reference:arachnids,181; classtype:shellcode-detect;
sid:648; rev:5;)
alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE
x86 NOOP"; content:"|61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61
61 61 61 61|"; classtype:shellcode-detect; sid:1394; rev:3;)
```

The NOOP instructions are used in buffer overflow and other types of attacks, to adjust the attack code to the right position in the stack/heap. As can be seen, all packets that contain a row of 21 a's - which is hex 61 or with the hex 90 signature corresponding binary data will generate a false positive. Again it must be determined from the packet logs if this is a true attack or not, or from Web logs, since most of the traffic here are to port 80. See http://www.whitehats.com/info/IDS181.

Row 10: CS WEBSERVER - external ftp traffic
Row 63: CS WEBSERVER - external ssh traffic
Like row 4 - I does not comment on this.

Row 11: [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC

Frequencies of servers with port numbers:

```
3072 64.235.110.34   saturn.izolnetworks.com 6667
2608 208.194.163.37  twisted.irctoo.net      6667
1684 205.160.101.121 irc.rma.edu             6667
 833 155.207.19.204  egnatia4.ee.auth.gr     6667
   3 205.188.149.12  undernet.irc.aol.com    6667
   1 MY.NET.100.165                            21
   1 212.161.35.251  beethoven.kewl.org      6667
```

As can be seen there must be a single false positive relating to an internal FTP server. The servers are mostly the same as the ones in the other alerts.

XDCC means file transfer traffic - the protocol are called DCC. This might of course be illegal "warez", but it might as well e.g.be scientific data or similar. Actually the alert message says that a client that has XDCC capacity is used. But I think that is the case with most popular clients (e.g. Xchat for Linux) - so if I understand the alert correctly this means just that one of those clients are used, not that files are transferred.

I think the University should be more specific here, allowing IRC and alerting on all kinds of file transfer, or even the capacity to do that seem a bit strange.

Row 12: High port 65535 tcp - possible Red Worm - traffic
The Red Worm (also known as Adore) are a Linux worm attacking vulnerabilities in *rpc.statd, Bind, LPRng*, and *wuftpd 2.6*, installing a trojaned version of the log daemon klogd that listens on this port. This might be connected to the alerts in row 15 and 23 - see below. Now in most of the alerts the 65535 port is on the external address, with the MY.NET address using a four digit port that varies, so this seems to be an ephemeral port. This does not seem to fit the information on the worm, which mentions some addresses in China. None of the external addresses seem to be in China - see table at the end of this paragraph.

The Red Worm theory can be easily checked on the internal machines. This can be done by finding out if they are Linux machines:

There is also a trojan called RC1 or Remote Control that defaults to port 65535, written in Visual Basic. I was not able to find much information on it. Most of the external machines seem to have names that suggest client nodes with various ISP's.

In the listing below is the 10 most frequent external addresses:

```
     34 MY.NET.100.230 65535
      26 MY.NET.25.69 65535
      25 MY.NET.25.70 65535
      21 MY.NET.6.63 65535
      17 MY.NET.6.55 65535
      11 MY.NET.25.10 65535
       9 MY.NET.25.71 65535
       7 MY.NET.25.73 65535
       7 MY.NET.25.12 65535
       5 MY.NET.84.151 65535
       5 MY.NET.25.11 65535
       4 MY.NET.60.17 65535
       4 MY.NET.25.72 65535
       2 MY.NET.100.13 65535
```

This might suggest that RC1 or something similar has gotten popular for controlling home PC's and similar. I have heard that SubSeven has been used this way, but since RC1 is not very well known, this is a bit doubtful. And then again someone inside the university might be doing outgoing attacks installing RC1 or something similar.

Another possibility is that port 65535 is a custom assignment to something entirely different. After all 65535 are the uppermost possible port number, so it is a quite obvious choice.

Certainly this should be investigated more closely!

```
grep 'High port 65535 tcp - possible Red Worm'  alert5.corr2|dp_xtr_ipad
|awk '{print $1}' |grep -v MY.NET|sort|uniq -c |sort -nr|head -10|awk
'{print $2}'|hostn.pl
63.164.243.132    cblmdm63-164-243-132.buckeye-express.com
12.248.17.145     12-248-17-145.client.attbi.com
217.209.142.239   h239n2fls31o1008.telia.com
195.235.137.81    137081.radredford.tsai.es
219.24.36.2       YahooBB219024036002.bbtec.net
68.32.63.62       pcp01838610pcs.owngsm01.md.comcast.net
63.201.230.252    adsl-63-201-230-252.dsl.snfc21.pacbell.net
151.196.123.195   pool-151-196-123-195.balt.east.verizon.net
64.157.4.78       mta-v22.level3.mail.yahoo.com
80.68.244.3       relay2.hotbox.ru
```

Row 13: MY.NET.30.4 activity

```
2637 524
  92 80
  11 17300
   4 666
   4 32559
   3 22
   1 21
```

Like in rows 4,8 and 10 this seems to be a general alert on a certain address. This does not seem a good idea to me, since either you get alerts on allowed traffic, or you could have blocked the traffic in a firewall or server access control list. The list of destination ports seems a bit like those of MY.NET.30.4, so obviously these are two similar servers:

Row 14: [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC
The IRC talkers involved here are:

```
1617 206.167.75.78      cricri.qeast.net
 146 129.143.67.242     irc.belwue.de
  82 206.252.192.195    irc-1.stealth.net
  19 213.186.35.9       ns336.ovh.net

These two has 1845 of the 1863 alerts
1617 MY.NET.84.228
 228 MY.NET.153.113
```

Sdbot is an IRC bot that is made to be installed as a trojan on Windows machines. These can then be controlled over an IRC channel to do IRC flooding and other more general trojan activities. MY.NET.84.228 and MY.NET.153.113 should be checked to see if this is the case.

Rows 15 and 23: connect to 515 from inside/outside
Todd Beardsley [38] commented thoroughly on this. There are several vulnerabilities on this service, Unix print. Among them is the Linux worm Adore in 2001, which I mentioned in the row 12 comment above. So this might suggest Adore invasion, but it does not fit the row 12 alert well as mentioned. Besides, Adore does not seem to be very persistent on the Internet, and should be uncommon in late 2002.

These rules are of doubtful use however, since they record all access, most of which will be ordinary printing. So these two alerts are certainly mostly false positives.

Better than alerting on port 515 use is to disallow printing across the network boundary. The University should then be less vulnerable to print daemon vulnerabilities, but I don't know if this is a possible policy. I comment more on this in the "Conclusions" subsection.

Note however that the alerts here involve just one pair of addresses for each alert. Probably this might be special cases that has permission to do this and that this is not reflected in the NIDS configuration. This should be easily dealt with in a firewall.

Row 16: External RPC call
The first alert message is unhelpful as regards what kind of RPC this is - and what it alerts on. But the destination port number is 111, which is assigned to the portmapper of Sun RPC. Actually this looks mostly like a portmapper scan from just these four addresses:

```
1095 211.114.9.211 -- NATIONAL CANCER CENTER, Korea
 193 195.13.253.73 -- Maris Sprancis, Latvia
  91 208.177.28.36 w036.z208177028.mia-fl.dsl.cnc.net XO
Communications, Virgina US
  37 211.22.153.30 mail.sun9am.com.tw Chunghwa Telecom Co, Taiwan

From the scan.* files:

1089 211.114.9.211
 193 195.13.253.73
  91 208.177.28.36
  37 211.22.153.30
   1 MY.NET.162.67
```

As can be seen these events are also reflected in the scan files, with almost the same counts. I see no reason to double report on this, so this rule should be not be needed.

Rows 17 and 39: ICMP/TCP SRC and DST outside network
The placement of the IDS sensor decides how to interpret this. It could reflect that some of these outside networks are routed through the University. Below is the Whois OrgName of the 5 most frequent talking pairs - they seems to have nothing to do with the University:

```
62 172.137.244.113 209.126.216.200
        America Online -> California Regional Internet, Inc.
27 66.93.118.119 66.93.118.118
        Speakeasy Network, Seattle -> Speakeasy Network, Seattle
27 172.145.252.133 65.95.240.171
        America Online -> Bell Canada
25 192.168.1.100 211.234.116.253
        (a private network) -> Asia Pacific Network Information
Centre
25 169.254.101.152 205.188.146.146
           IANA -> America Online
```

The second is strange - a provider in Seattle routes through the University. But of course, my knowledge of ISP geography (and the University's) is a bit limited. Besides, some of these addresses are from the private block 192.168 which should never be allowed outside their LAN. This might suggest that the University should reverse its routing policy. But it might also be because the NIDS configuration does not accurately reflect actual routing policy, and should be updated.

Rows 18 and 20: IDS552/web-iis_IIS ISAPI Overflow ida nosize (internal and external)
Donald Gregory [43] comments on this in his practical: "The ISAPI Overflow vulnerability is exploited by a malicious URL that can cause a buffer overflow in the idq.dll library used by the IIS servers". He goes on to say that the machines should be checked against Code Red but also that it may be false positives.

There should however be better signatures to discover Code Red - here is the default snort rule to discover CodeRed v.2 - this alerts on root.exe access:

```
snort-2.0.0/rules/web-iis.rules:alert tcp $EXTERNAL_NET any ->
$HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS CodeRed v2 root.exe access";
flow:to_server,established; uricontent:"/root.exe"; nocase;
```

The count here however seem to small to be explained by CodeRed infection, which will start massive outgoing attacks. This could either be some kind of false positives, or University addresses are used for large numbers of attacks. Packet logs should be examined to decide what is the case.

Row 19: Null scan!
Row 48: SYN-FIN scan!
I see no reason to have a rule for null scans and syn-fin scans, when they will be reported by the spp_portscan preprosessor, and in the scan files. Note that Snort changes most of the port numbers here to 0, this is a peculiarity of Snort when there are abnormalities in the header, and does not mean a port 0-0 scan. See the comment on the OOS files.

Row 21: NMAP TCP ping!
The nmap TCP ping is described in the nmap man page. It sends a TCP ACK package to a server - if it is up it will respond with a RST if the service is down, and a SYN-ACK if it is up. This is done with the -PT commando line option.

Below are the ten most common port combinations in the alert file.

```
    166 80 53
    138 80 80
     96 53 53
     84 80 4662
     31 81 80
     28 80 200
     28 80 143
     23 80 51200
     22 80 25
     10 80 50871
```

nmap defaults to destination port 80, but uses a 4-digit variable source port if you don't use the -g option to force a different port. This might indicate that the most frequent events here are something different, e.g. some kind of scan which uses a low port number.

If the rule just checks for ACK without corresponding SYN's, false positives will easily be generated e.g. when the SYN package is lost because of packet loss or for other reasons. I see no good reason for this rule, since nmap probes and scans should be alerted on by other rules.

Row 22: High port 65535 udp - possible Red Worm - traffic
Les Gordon has analysed this alert. He thinks this might be false positives caused by the AFS/Rx protocol. This protocol seems to be using various high ports. I found no other references to port 65535 being much used. If this is right the University should consider the security consequences of allowing this from outside the University network.

Row 24: NIMDA - Attempt to execute cmd from campus host
I have already commented on Nimda attacks in the row 1 comment. Les Gordon mentions false positives, but this depends on the signatures. I think this is avoidable - the message here says nothing about which command that was attempted.The signature "cmd.exe?/c+dir" should be quite good. Even better is to check for the string ".%255c." as mentioned.

```
   318 MY.NET.69.145
    85 MY.NET.97.61
    79 MY.NET.97.192
     1 MY.NET.97.39
     1 MY.NET.70.147
     1 MY.NET.132.45
     1 MY.NET.132.42
     1 MY.NET.10.177
```

Note also that there is a full record of a successful Nimda attack on MY.NET.69.145 in the scan files. This is further commented in the Scan subsection. At the left is a count of the sources for this alert - the three first addresses should be checked for Nimda.

Row 25 and 46: SUNRPC highport access/ Attempted Sun RPC high port access
These two rules must report on the same events, but they give different results! This should be investigated - an explanation might be that the one is for UDP and the other for TCP. Both is used by Sun RPC.

This is a quite common type of attack. RPC should be blocked in the firewalls when it is enabled on the servers. Often it is better to turn off RPC, but this depends on how they are used.

Row 26: Possible trojan server activity
This must alert on the port 27374, which are used by the trojans *li0n, Ramen, Seeker, SubSeven,* and *The Saint.* Below is a list of all the port combinations in the alert. As can be seen the most common events have the ports *25, 80, 110, 443, 445*. These are most probably false positives caused by Snort confusing source and destination. These addresses have second port numbers that is not much used (but this should be checked against what is common at the University). They should therefore be checked for trojans:

MY.NET.113.4, MY.NET.60.11, MY.NET.113.4, MY.NET.150.133, MY.NET.113.4, MY.NET.132.35, MY.NET.60.11

```
  89 27374 25              1 445 27374
  60 80 27374              1 34261 27374
  60 25 27374              1 27374 63598
  54 27374 80              1 27374 4616
  27 27374 110             1 27374 4461
  19 443 27374             1 27374 445
  19 27374 443             1 27374 4286
  17 110 27374             1 27374 4215
   6 1214 27374            1 27374 3782
   3 27374 3690            1 27374 3436
   3 27374 1214            1 27374 2507
   2 3690 27374            1 27374 2018
   2 27374 34261           1 27374 1765
   2 27374 2216
```

Row 27: SMB C access
This must alert on file sharing of the C: share. Generally it is not wise to allow file sharing to the world. This will inevitably lead to infection of some kind. And then sharing of the local disk on workstations will compromise all kinds of personal data. The University should block this, rather than just alerting on it.

Row 28: EXPLOIT x86 stealth noop
Here is the corresponding Snort 2.0.0 default rule fragment: *content: "|eb 02 eb 02 eb 02|" (SID 651)*. I assume the custom rule is similar. As can be seen false positives can be caused even more easily than by the rule in row 9 - her it is just eight characters to match. The most frequent pair here - *MY.NET.163.143* and *129.165.254.6 (g0acg01u.ecs.nasa.gov)* has the varying high port pairs shown in the listing on the left.

```
     55 56389 36584
     36 58298 36590
     32 55498 36582
     19 57652 36588
     18 58688 36569
     16 55911 36567
     16 48023 36561
     14 53408 36565
     13 57036 36586
      8 60965 36571
      2 57300 36602
```

This might be part of file transfer sessions - e.g with passive FTP, where each pair represent a transfer session. The time stamps confirm this, as the port pairs occur roughly in the same time windows.

Row 29: Incomplete Packet Fragments Discarded
I don't know enough to comment - I should have seen the packets. The port number here is listed as 0 by Snort - this is because the packets are fragments, which have no port numbers.

Row 30: SNMP public access
SNMP access should bee blocked from the outside. Using the read community "public" is usually unwise, because this will expose the University to information leak, and even compromise. But this might be a policy question. Some routers are left open to give info to peering ISP's.

Row 31: NIMDA - Attempt to execute root from campus host

```
     103 MY.NET.69.145
      19 MY.NET.97.192
      14 MY.NET.97.61
```

A count of the alerts seems to indicate three infected machines, and they should be checked. See listing on the left. Note that *MY.NET.69.145* is also found by correlating the top scanners from the scan files with the alerts - see comment on row 24 and in the Portscan subsection:

Row 32: TFTP - Internal TCP connection to external tftp server
Here are the server addresses for these alerts:

```
82 198.64.149.228 ---               Verio, Inc, CO, USA
50 198.173.255.237 supershe.tempdomainname.com, Verio, Inc, CO, USA
1  140.239.42.26   c0026.harvard.net, Allegiance Telecom Companies
Worldwide
```

The MY.NET addresses vary with none of the addresses more than twice. Note that most of the alerts are inverted with the TCP server address first. This is as mentioned before quite common in Snort alerts, and reflect return traffic from the server. The alerts seem to concern some special service from Verio, Inc , an ISP and domain name provider. But I was unable to find out what it is. TFTP normally uses UDP, so this is probably a different protocol. Les Gordon has commented on similar alerts - but from external sources. He recommends more detailed analysis, since the traffic seems strange.

Row 33: FTP passwd attempt
It is hard to understand what is meant here - could it be *failed* attempts? Just one MY.NET address is involved - *MY.NET.24.47*. I find nothing to comment without knowing more.

Row 34: TFTP - Internal UDP connection to external tftp server
Note that Nimda uses tftp/UDP , so this might be a sign of infection when scan activity follows. Of the addresses here *MY.NET.83.69* and *MY.NET.69.145* is obviously infected with Nimda, as mentioned in the Portscans subsection. The *MY.NET.1.3, MY.NET.1.4* and *MY.NET.1.5* alerts has 53 or 123 as the other port, so this might be DNS and NTP requests with unusual source ports - if these are DNS servers. The other addresses - *MY.NET.114.110, MY.NET.117.10, MY.NET.150.203, MY.NET.152.184, MY.NET.152.246, MY.NET.152.250, MY.NET.153.105, MY.NET.153.195, MY.NET.5.92, MY.NET.70.134* - should be checked for Nimda if they are Windows boxes.

Row 35: EXPLOIT x86 setuid 0
Row 41: EXPLOIT x86 setgid 0
The setuid 0 alert corresponds to SID 650 "SHELLCODE x86 setuid" and the setgid 0 to SID 649 "SHELLCODE x86 setgid 0", which according to the Snort doc page is part of several buffer overflow attacks. The content part of the two rules:
content: "|b0b5 cd80|"; and *content: "|b017 cd80|"*

The rules have lots of false positives. I assume this is the case for the University rule too, so more analysis of the packets is needed to determine the exact attack. See also:

http://www.whitehats.com/info/IDS436 A peculiarity here is that 17 of the alerts have port 4000 as one of the ports.

Row 36: EXPLOIT NTPDX buffer overflow

```
28 63.250.195.10 MY.NET.150.203
 4 198.64.140.205 MY.NET.97.44
 3 206.204.200.108 MY.NET.18.22
 2 208.153.50.192 MY.NET.151.115
 1 63.250.195.10 MY.NET.69.249
 1 63.250.195.10 MY.NET.117.10
 1 212.113.174.194 MY.NET.84.198
 1 208.172.128.163 MY.NET.74.247
 1 204.118.192.170 MY.NET.151.115
 1 198.64.140.205 MY.NET.97.63
 1 12.129.72.165 MY.NET.97.22
```

This alerts on a buffer overflow of the ntpd time protocol daemon (NTPX is the name of the published exploit) See: *htttp://www.securityfocus.com/archive/1/174011*

On the left are all the talking pairs - the My.Net servers should be checked if they listen to port 123 and have a vulnerable NTPD. NTP might be protected by key authentication that ensures that servers are who they should be- see [56]. This might not be good enough against buffer overflow attacks, since they may be performed before authentication. Of course prompt patching when vulnerabilities are made known is important.

Row 37: Tiny Fragments - Possible Hostile Activity
This might be part of a fragmentation attack, but several other possibilities exist. I have to see the packets to comment.

Row 38 and 42: Notify Brian B. 3.54/3.56 tcp
MY.NET.3.54 and 3.56 seem to be web servers with traffic to port 80. Like with the CS Web server it is not possible to say much here, since I do not know the background for this. But in all these cases it would be better to use a firewall or a web server access list to control access.

Row 40: IRC evil - running XDCC
Row 50: [UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot
Row 53: [UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected
Row 55: [UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot
There are all together five alerts involving XDCC activity. They must be different since they give different results, but is not clear to me what the difference is. The alert in row 11 with 8201 alerts is on a different level than the other with a total count of 56. See the comment on row 11 for an explanation of XDCC. I have put the four less frequent alerts together to compare them - here is a table of them:

Count of the four alerts	35 IRC evil - running XDCC 10 [UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot 8 [UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected. 3 [UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot
Top 5 talkers of all alerts	17 MY.NET.74.216 212.161.35.251 10 MY.NET.198.221 205.188.149.12 8 MY.NET.80.209 66.207.164.23 6 212.161.35.251 MY.NET.74.216 3 209.126.191.153 MY.NET.83.48
All involved MY.NET servers	23 MY.NET.74.216 10 MY.NET.198.221 9 MY.NET.80.209 8 MY.NET.17.48

	4 MY.NET.83.48 1 MY.NET.97.61 1 MY.NET.82.41
IRC evil - running XDCC	17 MY.NET.74.216 212.161.35.251 10 MY.NET.198.221 205.188.149.12 8 MY.NET.80.209 66.207.164.23
[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot	3 209.126.191.153 MY.NET.83.48 2 66.118.185.29 MY.NET.17.48 1 69.50.166.81 MY.NET.17.48 1 66.118.164.157 MY.NET.17.48 1 65.86.165.180 MY.NET.17.48 1 216.152.64.155 MY.NET.17.48 1 210.15.254.126 MY.NET.17.48
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected	6 212.161.35.251 MY.NET.74.216 1 66.207.164.23 MY.NET.80.209 1 64.62.96.34 MY.NET.83.48
[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	1 24.94.220.84 MY.NET.97.61 1 216.65.55.82 MY.NET.17.48 1 216.194.70.9 MY.NET.82.4

Table 8. IRC alert talking pairs.

There is just one of the MY.NET address here that occurs in non-IRC alerts - *MY.NET.97.61.* It is in the Nimda alert in row 20 and 31, and suspected of Nimda infection. Since the policy behind these four alerts seems unclear, both the alerts and the policy should be re-evaluated. Questions like is XDCC illegal, how to detect illegal bots - and should all bots be banned - should be answered.

Row 43: NETBIOS NT NULL session

This is mostly part of normal Windows communication. Netbios (or more properly SMB) is a vector for all kinds of security problems - but most alerts are false positives. The University have to live with this if it wants to allow SMB communication with the world, but to day this also means having irregular worm crises and other security problems. It would be better to block Netbios at the borders, and use methods that give better protection when allowing access to the University network.

Row 44: RFB - Possible WinVNC - 010708-1

```
7 209.240.190.63 5900
4 141.156.18.91  5900
2 MY.NET.70.225  5900
2 MY.NET.111.188 5901
1 MY.NET.178.31  5900
1 MY.NET.111.51 5900)
```

Here RFB means Remote Frame Buffer, and VNC means Virtual Network Computing. The last is a freeware remote console (or remote admin) system for several platforms, and RFB is the name of the underlying protocol. WinVNC is a Windows server for VNC. I was unable to find a reference to the code *010708-1.* Note that here Snort inverts the connections - alerting on both directions. VNC servers in the alerts has port number: 5900 and 5901 - the last digit corresponds to different consoles.

Like other remote admin systems, VNC has several security problems, like unencrypted password, a couple of cases of buffer overflow vulnerabilities and so on. See [46]. An important improvement of security is mentioned in this reference - tunnelling through SSH. Never the less I am not sure what the reason for this alert is.

Row 45: DDOS shaft client to handler

The corresponding default Snort rule here triggers on port *20432*. This port is also the second port in the alerts here. The first port here is either 25 or 80, so these alerts is probably false positives triggered by ephemeral port 20432 in return packets. See part 1 in this practical (row 29 in the alerts - Table 1) for a comment on this alert.

Row 47: Traffic from port 53 to port 123
It is hard to tell what this is without knowing more. There must be a reason for the rule. Sometimes one sees DNS traffic with low source ports like this. A possible explanation is some sort of spoofed source reverse traffic probing. Both 53/udp (DNS) and 123/udp (Network Time Protocol) tend to be open in firewalls. But I am not sure what can be gained by this compared to more ordinary scanning. The address in question *64.125.197.7 corp-gw.viva.yellowbrix.net* does not answer DNS requests so it is no DNS server. The DNS name makes it look like a private subscription node, possibly a dynamic address.

Row 49: External FTP to HelpDesk MY.NET.70.50
Row 51: External FTP to HelpDesk MY.NET.70.49
Row 61: External FTP to HelpDesk MY.NET.53.29
Like other similar alerts I don't now enough to comment.

Row 52: TCP SMTP Source Port traffic
I don't know enough to comment - if these are mail servers, this might be Snort mixing source and destination. This is probably meant to discover "reflected" attacks on mail servers, but I doubt this is effective.

Row 54: [UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan
Address-port combinations of the four alerts are in the listing below. *K\:line* is similar to the Kill command. It is unclear to my why this is a sign of a trojan, since ordinary users can be K\:lined by server op's. Besides the port 6969 is not usually used for IRC, so it might be something different. *www.simovits.com* give a long list of trojans on this port, some of which is IRC related - so the second address pair here might be a sign of MY.NET addresses operating a trojan on 210.146.253.10 after all. This should be checked.

```
205.188.149.12:6667 -> MY.NET.60.11:22339
210.146.253.10:6969 -> MY.NET.97.186:1100
210.146.253.10:6969 -> MY.NET.97.186:1148
210.146.253.10:6969 -> MY.NET.97.186:1226
```

Row 56: Probable NMAP fingerprint attempt
Address-port combinations of the four alerts are in the listing below. For a description of how nmap do fingerprinting - see [45]. I find alerts like this of little use if one has not meant to act on it, it might be stored to get a timeline of a full attack possibly - but a mere fingerprinting will most often be done without further attack. I think following up cases like that is far too much work on a busy site.

```
63.251.52.75 16409 MY.NET.97.59 28945
63.251.52.75 28640 MY.NET.97.59 48162
61.48.209.23 21322 MY.NET.25.73 18724
```

Row 57: DDOS mstream handler to client
See comment on this in this Practical part 1. If the signature is the same as in the default Snort rule, the chance is small that this is the real thing. Here are the two alerts:

```
07/06-03:03:04.321149  [**] DDOS mstream handler to client [**]
MY.NET.6.55:15104 -> 207.69.200.154:25
07/06-03:03:04.338010  [**] DDOS mstream handler to client [**]
MY.NET.6.55:15104 -> 207.69.200.154:25
```

In all probability these are ordinary mail traffic which coincidentally contain an ">" and use ephemeral port 15104. See part 1 in this practical (row 30 in the alerts - Table 1) for a comment on this alert.

Row 58: TFTP - External UDP connection to internal tftp server
This should be checked, normally TFTP should only be used on closed networks for downloading router images and the like (or even better not at all). The involved address is *MY.NET.152.246.*

Row 59: FTP .forward
The server must be checked - no FTP server should allow this from a user home directory - but of course the string ".forward" can be something else. The involved address is *MY.NET.99.5.*

Row 60: Fragmentation Overflow Attack
The involved pair here is *151.196.121.201:0 -> MY.NET.11.4:0* . This is probably an alert from the snort preprossesor frag2 or similar, warning against a Teardrop type of attack with overlapping offsets. This should normally be ignored, since modern stacks should be proof against this. Alternatively a warning can be given to those responsible for user support, in case there is a complaint of malfunction of the reported address. A good treatment of two Teardrop variants is in [8] - taken from Mark Cooper and Alva Veach.

Row 62: EXPLOIT x86 NOPS
It is hard to guess in what way the rule here differs from "EXPLOIT x86 NO" - see above, row 9. Therefore I give no comment here, but the talking pair is *205.188.72.22:5190 -> MY.NET.98.66:3037*

Row 563: Back Orifice
This should of course be investigated - the talking pair is *198.64.140.205:20880 -> MY.NET.97.44:31337.*

Port scans

Perhaps controversially, I tend not to put too much emphasis on incoming port scans, seeing them as normal background noise of the network. I feel it is unpractical to log this with a signature based NIDS - a firewall will do this much better (if you have one in position that is). See also comment in this practical, part 1. Outgoing port scans might be a bit more interesting, because they may be a sign of compromised boxes. To look for this is only effective however if it does not drown in port scans done by curious students.

A somewhat special note about the scan files: There are no MY.NET addresses in them. I guess that the 130.85 addresses are what have been changed to MY.NET, if that is the case then it is clear where the logs analysed here come from! I think this cannot be the intention, so if GIAC want to fix this, I give permission to delete this paragraph before publishing my practical. In the following I have changed the two doubtful octets to MY.NET.

```
nohup awk '{OFS=":";print $4,$6,$7,$8}' scans.03070*| awk -F: '{print
$1,$4,$5,$6}' > file1
nohup sort -T . file1 > file2 &
nohup uniq -c file2 > file3&
nohup sort -nr file3>file4&
```

The scan files was to large for sorting and selecting in one step on my humble test machine. I did this in four steps using *nohup* to avoid broken jobs when logging off or losing connection. Note that there is a discrepancy between the scan files and the spp_portscans alerts. E.g. *MY.NET.114.45*, which has the following count of the two types of files. This is of course because of the missing alert file on July 4.

```
awk '/End of portscan/{print $8,$11}' alert.03070*|grep
MY.NET.114.45|perl -ne '/hosts\((\d+)\)/;$s=$s+$1;print "$s\n"'
......
13082447

awk '/130.85.114.45/{s=s+$1;print s}' file4 |tail -1
17646127

counting the days corresponding til the alert files:
nohup awk '{print $4}' scans.030702 scans.030703 scans.030705
scans.030706 scans.030707 | grep '130\.85\.114\.45' > file.114_45 &
wc -l file.114_45
13994889 file.114 45
```

Top 15 outgoing port scanners:

```
17646066 MY.NET.114.45 80 SYN
 2885272 MY.NET.1.3 53 UDP
 2162840 MY.NET.69.145 80 SYN
  898302 MY.NET.1.4 53 UDP
  287702 MY.NET.100.230 25 SYN
  222945 MY.NET.97.15 137 UDP
  161498 MY.NET.97.80 137 UDP
  114297 MY.NET.97.23 137 UDP
  111871 MY.NET.97.240 137 UDP
  100850 MY.NET.83.69 4662 SYN
   96327 MY.NET.99.48 41170 UDP
   93594 MY.NET.97.170 137 UDP
   88720 MY.NET.97.51 137 UDP
   81887 MY.NET.97.188 137 UDP
   77810 MY.NET.97.73 80 SYN
```

The port numbers in the listing at the left are destination ports.The two port 53 UDP lines seem to imply that the two addresses are DNS servers.
The addresses here should be checked for compromise, but this must be corrected for if some of them are web proxies or equivalent, for then it might be false positives.

Observe the one line with port 4662 here - see the next section about this. Observe also the line with destination port 41170. This seems to be related to some file sharing protocol UDF, which might be Gnutella-related. The only reference I found on this was in German, see [44]. Here is room for more investigation.

Correlating alerts and top 15 outgoing scans

MY.NET.114.45	1 Null scan! (port 139) 134.22.118.48
MY.NET.1.3	36 NMAP TCP ping! 63.211.17.228:80 -> MY.NET.1.3:53 36 NMAP TCP ping! 63.211.17.228:53 -> MY.NET.1.3:53 34 NMAP TCP ping! 64.152.70.68:80 -> MY.NET.1.3:53 34 NMAP TCP ping! 64.152.70.68:53 -> MY.NET.1.3:53 11 Traffic from port 53 to port 123 64.125.197.7:53 -> MY.NET.1.3:123

	(several more NMAP TCP ping)
MY.NET.69.145	37 spp_http_decode: IIS Unicode attack detected 80.58.5.109 -> MY.NET.69.145:80 7 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize MY.NET.69.145 -> 130.126.118.103:80 6 TFTP - Internal UDP connection to external tftp server MY.NET.69.145 -> 130.13.153.74:69 6 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize MY.NET.69.145 -> 130.63.94.58:80 6 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize MY.NET.69.145 -> 130.63.234.223:80 (several more alerts...) or just the alerts on this address: 539 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 318 NIMDA - Attempt to execute cmd from campus host 103 NIMDA - Attempt to execute root from campus host 41 spp_http_decode: IIS Unicode attack detected 6 TFTP - Internal UDP connection to external tftp server
MY.NET.1.4	13 NMAP TCP ping! 64.152.70.68:80 -> MY.NET.1.4:53 13 NMAP TCP ping! 63.211.17.228:80 -> MY.NET.1.4:53 11 NMAP TCP ping! 63.211.17.228:53 -> MY.NET.1.4:53 10 NMAP TCP ping! 64.152.70.68:53 -> MY.NET.1.4:53 5 NMAP TCP ping! 193.144.127.9:80 -> MY.NET.1.4:53 (several more NMAP TCP ping)
MY.NET.100.230	72 Queso fingerprint 63.71.152.2 -> MY.NET.100.230:113 20 High port 65535 tcp - possible Red Worm - traffic MY.NET.100.230 -> 129.6.59.2:25 11 Queso fingerprint 216.65.124.73 -> MY.NET.100.230:25 6 Queso fingerprint 209.47.197.14 -> MY.NET.100.230:25 5 High port 65535 tcp - possible Red Worm - traffic MY.NET.100.230 -> 147.91.242.1:25 (several more of the same kind)
MY.NET.97.15	537 spp_http_decode: IIS Unicode attack detected MY.NET.97.154 -> 202.103.69.100:80 202 spp_http_decode: IIS Unicode attack detected MY.NET.97.15 -> 64.12.50.217:80 173 spp_http_decode: IIS Unicode attack detected MY.NET.97.15 -> 218.30.12.58:80 36 spp_http_decode: IIS Unicode attack detected MY.NET.97.15 -> 207.200.86.66:80 21 spp_http_decode: IIS Unicode attack detected MY.NET.97.15 -> 64.12.48.217:80
MY.NET.97.80	75 spp_http_decode: IIS Unicode attack detected MY.NET.97.80 -> 64.12.39.89:80 49 spp_http_decode: IIS Unicode attack detected MY.NET.97.80 -> 218.30.12.58:80 36 spp_http_decode: IIS Unicode attack detected MY.NET.97.80 -> 205.188.139.152:80 27 spp_http_decode: IIS Unicode attack detected MY.NET.97.80 -> 207.200.86.65:80 9 spp_http_decode: IIS Unicode attack detected MY.NET.97.80 -> 64.12.188.120:80
MY.NET.97.23	215 spp_http_decode: CGI Null Byte attack detected MY.NET.97.238 -> 217.174.99.5:80 141 spp_http_decode: IIS Unicode attack detected MY.NET.97.23 -> 211.233.32.11:80

	132 spp_http_decode: IIS Unicode attack detected MY.NET.97.235 -> 4.35.253.46:80 116 spp_http_decode: IIS Unicode attack detected MY.NET.97.235 -> 162.33.122.171:80 102 spp_http_decode: IIS Unicode attack detected MY.NET.97.232 -> 211.233.29.53:80
MY.NET.97.240	34 spp_http_decode: IIS Unicode attack detected MY.NET.97.240 -> 199.244.218.42:80 3 spp_http_decode: IIS Unicode attack detected MY.NET.97.240 -> 64.12.50.217:80 1 spp_http_decode: IIS Unicode attack detected MY.NET.97.240 -> 149.174.32.135:80
MY.NET.83.69	30 Null scan! 61.64.177.231 -> MY.NET.83.69:4662 8 TFTP - Internal UDP connection to external tftp server MY.NET.83.69 -> 216.17.103.14:69 4 NMAP TCP ping! 140.138.7.2 -> MY.NET.83.69:7701 2 spp_http_decode: IIS Unicode attack detected MY.NET.83.69 -> 202.107.53.23:80 2 NMAP TCP ping! 195.77.24.2 -> MY.NET.83.69:7701
MY.NET.99.48	7 Tiny Fragments - Possible Hostile Activity 24.242.101.56 -> MY.NET.99.48
MY.NET.97.170	3 spp_http_decode: IIS Unicode attack detected MY.NET.97.170 -> 211.233.79.208:80
MY.NET.97.51	56 spp_http_decode: IIS Unicode attack detected MY.NET.97.51 -> 218.30.12.58:80 2 spp_http_decode: IIS Unicode attack detected MY.NET.97.51 -> 216.136.227.14:80 1 spp_http_decode: IIS Unicode attack detected MY.NET.97.51 -> 209.249.64.242:80
MY.NET.97.188	3273 spp_http_decode: IIS Unicode attack detected MY.NET.97.188 -> 202.103.69.100:80 478 spp_http_decode: IIS Unicode attack detected MY.NET.97.188 -> 218.30.12.75:80 46 spp_http_decode: IIS Unicode attack detected MY.NET.97.188 -> 218.30.12.58:80 30 spp_http_decode: IIS Unicode attack detected MY.NET.97.188 -> 207.188.7.118:80 16 spp_http_decode: IIS Unicode attack detected MY.NET.97.188 -> 65.54.244.250:80
MY.NET.97.73	12 spp_http_decode: IIS Unicode attack detected MY.NET.97.73 -> 211.32.117.26:80 3 spp_http_decode: IIS Unicode attack detected MY.NET.97.73 -> 211.233.28.127:80 3 spp_http_decode: IIS Unicode attack detected MY.NET.97.73 -> 194.67.57.50:80 1 EXPLOIT x86 NOOP 207.46.177.148 -> MY.NET.97.73:3121

Table 9. Correlating top 15 outgoing scanners with the alert files. 5 most frequent alerts.

Remarkably the top outgoing scanner here has just one alert. It is hard to guess the reason for this without knowing what kind of address this is.The two DNS servers MY.NET.1.3 and MY.NET.1.4 are well represented here, but I assume this to be false positives, as I also comment above. Further there is good indication here of Nimda infection for the following addresses, confirming the suspicions I mention in the comment to row 2: *MY.NET.69.145*, *MY.NET.97.15, MY.NET.97.80, MY.NET.97.2,MY.NET.97.240, MY.NET.83.69, MY.NET.97.170, MY.NET.97.5, MY.NET.97.188*, *MY.NET.97.73.* But probably there are many more, so there should be a campaign against Nimda. Note that with *MY.NET.69.145* and *MY.NET.83.69* even Nimda's tftp download are shown.

Out of spec files

To analyse the OOS files, I normalized them to a single line format with:

```
egrep -v '^$' OOS_Report_2003_07_0x_xxxx|perl -ne 'if (/^=+/) {print
"$l\n"} elsif (/^\d\d\/\d\d-/){chop;$l=$_} else  {chop;$l="$l $_"} '
> oosnorm.03070x
```

The lists at the bottom of the page gives frequency count of the addresses in the OOS files - using the usual sort-uniq-sort method. There is just one address behind the *****SF* combination (we ignore the single event).

```
[kbjo@minestrone anthis]$ grep '\*\*\*\*\*\*SF' oosnorm.03070*|xtr_ipad
|awk '{print $1}'|sort|uniq -c|sort -nr|head -20
  24758 142.26.120.7
      1 213.97.9.122
```

This is a typical SYN-FIN sweep for FTP servers (with src port set to 21), but it is not recorded in the spp_portscan alerts, maybe because the threshold is set to low? The address 142.26.120.7 is owned by British Columbia Systems Corporation, but have no reverse DNS record

```
  Top 10 OOS talkers
   1436 168.226.117.32 MY.NET.112.196
    664 200.51.212.184 MY.NET.112.196
    572 168.226.118.34 MY.NET.112.196
    418 80.143.95.179 MY.NET.112.196
    331 168.226.117.108 Y.NET.112.196
    265 212.114.238.229 MY.NET.189.41
    264 67.119.233.217 MY.NET.12.4
    216 212.114.237.186 MY.NET.189.41
    210 12.255.198.216 MY.NET.24.44
    193 200.51.212.120 MY.NET.112.196
```

```
Top 5 flag combinations:
  24759 ******SF
  15581 12****S*
    625 ********
    307 ****P***
    123 12***R**

The other combinations have
frequencies of 2 and 1, so seems to
be singular events I won't go into
here.
```

```
Top 10 OOS sources
  24758 142.26.120.7
   1436 168.226.117.32
    689 213.186.35.9
    664 200.51.212.184
    572 168.226.118.34
    528 67.119.233.217
    418 80.143.95.179
    360 209.47.197.12
    345 209.47.197.14
    334 216.95.201.21
```

```
Top 10 OOS destinations
   4065 MY.NET.112.196
   1247 MY.NET.25.70
   1215 MY.NET.25.72
   1212 MY.NET.25.69
   1157 MY.NET.25.73
   1150 MY.NET.25.71
   1056 MY.NET.24.44
    843 MY.NET.189.41
    364 MY.NET.24.34
    327 MY.NET.100.165
```

.

Top 10 external scanners	Top 10 dest ports	MY.NET addresses receiving 4662 traffic	MY.NET addresses receiving SMTP traffic
1436 168.226.117.32	7333 25	4064 MY.NET.112.196	1244 MY.NET.25.70
689 213.186.35.9	5127 4662	842 MY.NET.189.41	1209 MY.NET.25.72
664 200.51.212.184	2117 80	192 MY.NET.111.197	1206 MY.NET.25.69
572 168.226.118.34	158 113	27 MY.NET.112.195	1155 MY.NET.25.73
418 80.143.95.179	96 81	1 MY.NET.97.207	1141 MY.NET.25.71
360 209.47.197.12	94 8080	1 MY.NET.97.114	223 MY.NET.24.23
345 209.47.197.14	75 8888		207 MY.NET.6.47
334 216.95.201.21	74 8081		206 MY.NET.24.21
331 168.226.117.108	74 8001		138 MY.NET.24.22
301 209.47.197.13	71 1214		etc.

The table above give a summary of 12****SF* activity (i.e. SYN packets with reserved bit 1 and 2 set). The port 25 (SMTP) events are directed to several MY.NET addresses, so might be scans and fingerprint attempts from the outside. I have singled out the port 4662 events since that might indicate internal addresses used by the eDonkey protocol to give out p-to-p file sharing material. I can't give a good reason for the reserved bits being used here so the MY.NET addresses involved (primarily MY.NET.112.196 on port 4662) should be inspected.

The 4662 port also involves the Queso fingerprint alert - Queso uses the two bits, so this might be the reason. But note that Extended Congestion Notification also uses the bits, so this issue must be seen as undecided. Checking on the above *MY.NET 4662* addresses showed that they all had corresponding *Queso fingerprint* alerts. See the comment to *row 6* in the alert table above. See also [57]

```
    264 MY.NET.12.4 110
     71 MY.NET.25.24 110
     69 MY.NET.25.23 110
     63 MY.NET.25.22 110
     61 MY.NET.25.21 110
     25 MY.NET.83.69 4662
     12 MY.NET.114.120 6699
      5 MY.NET.29.11 443
      5 MY.NET.114.45 3019
      4 MY.NET.97.61 1581
```

On the left is a count of ******** (no flags set) - the port 110 counts seems to correspond to null scans of the POP3 protocol from 67.119.233.217, a private subscription address with Pac Bell Internet Services. I see no reason to do anything with this without more correlating information about something wrong.

```
139 MY.NET.69.217 3456
 78 MY.NET.97.55 3393
 54 MY.NET.69.217 80
 11 MY.NET.132.50 16952
 10 MY.NET.150.220 1214
  9 MY.NET.83.69 4662
  3 MY.NET.113.4 1214
  2 MY.NET.111.34 1214
```

On the left is a count of ***P*** (PSH flag) - here I fail to see why these packets are out of spec, since the PSH flag is a normal part of TCP communication. I am certainly overlooking something. On the other hand, here the entire packet is copied to the logs in hex and character format, so it is possible to see the payload. In these the string "Kazaa" is in 198 packets, among them all the packets with "Kazaa" in them. Now Kazaa uses destination port 1214, so the two first lines here are a bit mysterious. Note that a Trojan -TerrorTrojan - uses port 3456. I did not find a reference on port3393. The two first addresses should be checked.

Left addresses	Right address
62 MY.NET.12.2 25 31 MY.NET.12.4 993 14 MY.NET.12.4 143 6 200.67.129.59 7864 2 MY.NET.12.4 110	11 172.193.199.14 1381 8 172.193.199.14 1261 8 172.193.156.113 1304 6 MY.NET.24.44 80 6 MY.NET.100.165 80

On the left is a count of 12***R** (ie. RST with reserved bit 1 and 2 set). As can be seen obviously some return traffic is represented here, with traffic to MY.NET.12.2 most numerous. This might possibly be a mail server. 993 is IMAP over SSL, 143 is IMAP2, 110 POP3. 1381, 1261, 1304 I found no references for. These packets might be some form of fingerprinting (by Queso?), but more investigation is needed to decide what these OOS packets really are.

Description of the analysis process

I have only the Snort alert, scan and OOS files to base my analysis on here, so it is somewhat limited. Mostly I base it on frequency counts of the three file types. This has the limitation that I might overlook important singular events. I have no overview of the University network, no packet dumps and no knowledge of overall security policy. This is in some respects worse than a blind penetration test, where you normally get permission to do port scans, and other forms of probing and reconnaissance. It is impossible in practice to fully evaluate singular events, but the event counts give a useful picture never the less.

So I have performed a count of the alert files, trying to deduce things from the alerts, as shown in detail in the comments to the alert count table 6 above. Then I have counted the port scans, and tried to correlate it to the alerts. At last I have done the same with the OOS files, also trying to correlate it to the alerts.

Conclusions: Defensive recommendations

From what I have found out from the logs here, there are several problems with the security of the University. First - below is a table summarizing machines that should be checked.

Address	**Action**	**Referred to in:**
MY.NET.69.145 MY.NET.97.15 MY.NET.97.80 MY.NET.97.23 MY.NET.97.240 MY.NET.83.69 MY.NET.97.170 MY.NET.97.51 MY.NET.97.188 MY.NET.97.73 MY.NET.97.192 MY.NET.97.61 MY.NET.114.110 MY.NET.117.10 MY.NET.150.203 MY.NET.152.184 MY.NET.152.246 MY.NET.152.250 MY.NET.153.105 MY.NET.153.195 MY.NET.5.92 MY.NET.70.134	Check for Nimda, disinfected and patched if necessary	Alert table row 2,31 and 34 Subsection on port scans
MY.NET.97.231 MY.NET.108.48 MY.NET.152.159 MY.NET.97.143 MY.NET.97.150 MY.NET.97.192 MY.NET.97.235 MY.NET.97.62 MY.NET.97.71 MY.NET.97.85	Check for trojan infection - gives IRC alert and uses port	Alert table row 7

MY.NET.97.211	7000. Note that MY.NET.97.192 also is a possible Nimda victim	
MY.NET.100.230 MY.NET.25.69 MY.NET.25.70 MY.NET.6.63 MY.NET.6.55 MY.NET.25.10 MY.NET.25.71 MY.NET.25.73 MY.NET.25.12 MY.NET.84.151 MY.NET.25.11 MY.NET.60.17 MY.NET.25.72 MY.NET.100.13	Check for Red Worm/Adore - first check if they use Linux. Also check for the RC1 trojan	Alert table row 12
MY.NET.84.228 MY.NET.153.113	Check for Sdbot	Alert table row 14
MY.NET.113.4 MY.NET.60.11 MY.NET.150.133 MY.NET.132.35	Check for the trojans li0n, Ramen, Seeker, SubSeven, and The Saint. Uses port 27374	Alert table row 26
MY.NET.150.203 MY.NET.97.44 MY.NET.18.22 MY.NET.151.115 MY.NET.69.249 MY.NET.117.10 MY.NET.84.198 MY.NET.74.247 MY.NET.97.63 MY.NET.97.22	Check for NTP compromise if they listen on port 123. Note that Windows are not among the vulnerable architectures	Alert table row 36
MY.NET.70.225 MY.NET.111.188 MY.NET.178.31 MY.NET.111.51	Check for VNC compromise	Alert table row 44
MY.NET.60.11 MY.NET.97.186	Check for trojan use in an IRC context	Alert table row 54
MY.NET.114.45	Massive scanning with this as source should be checked	Subsection on port scans
MY.NET.112.196 MY.NET.189.41 MY.NET.111.197 MY.NET.112.195 MY.NET.97.207 MY.NET.97.114	The addresses have eDonkey traffic with TCP reserved bits 0 and 1 set	Subsection on OOS files
MY.NET.69.217 MY.NET.97.55	Check for trojan use in connection with Kazaa traffic	Subsection on OOS files

Table 10. Summary of addresses that should be checked.

- There are massive port scans from the inside network - this might be a sign of compromise or of too lax policy towards student cracker activity. A closer look shows that much of this is due to a Nimda infection. This is no good sign of the state of security with the University, since the main Nimda attack was in the autumn 2001! Perhaps this is a problem of responsibility, since with a university it is hard to have a strict centralised management of IT services.

- Some of the more numerous alerts has no good explanation here, and must be investigated further by checking packet logs, server logs etc. I will especially mention
    - Row 3 in the alert table "CGI null byte attack"
    - Row 6 Queso fingerprint
    - Row 7 IRC user /kill detected
    - Row 9 (and less numerous row 28, 35,36,41 and 62) the EXPLOIT x86 alerts

These are most probably false positives, but possible compromise must be ruled out.

- There is some IRC traffic that generates alerts, as commented on in the alert subsection. The University should review its policy towards the use of IRC. If this is used for serious purposes a closed solution should be considered. By this I mean setting up dedicated IRC servers controlled by the University - where strict control of the traffic might be performed. If possible, one might cooperate with other server owners on the peering traffic between the servers. Then it may be possible to restrict problematic channels and practices.

All other use of IRC might be banned. I know this sounds very restrictive. But I think that the culture surrounding this service is unsound, and that the alternative is continuous problems with compromise within the University, and lack of control of what is done from the inside against others on the Internet.

- I think there should be cooperative security work in system and network administration where the NIDS should be a part of a wider picture. Now I don't know much about how this is done at the University, but I think the NIDS logs suggest things could be better here. Firewalls does not seem to be fully used to block undesirable traffic, and the worm attack might be a sign that security work could be better within server administration and user support.

- Some policies might be revised, e.g. the policy towards printing over the network. I see no reason to let outsiders print to the University printers. Employees and students who want to print, and do other activities from home computers and other external sources should be provided with encrypted and authenticated channels to the inside. The same must be said about the use of Microsoft SMB protocols and Sun RPC protocols outside networks protected by firewalls. Ports that should be blocked on border firewalls are SMB: 135-139/tcp and udp, 445/tcp, 111/tcp, 515/tcp and so on. Much better is a firewall policy of deny all in the bottom, and opening specific ports and address ranges that need to go trough.

- Policy-related use of NIDS might be problematic. If e.g. printing from the inside and out is not allowed, and alerts are generated like in row 15, what should then be done? A sanction like taking away user rights on the University network might be seen as excessive, and will be costly to execute if there are large numbers of people doing this. It is better to stop things in a firewall or similar, then nobody will be able to break the rules, and there will be relatively fewer cases of people demanding an exception from the policy rules to deal with.

- I see very little attention to p2p activities here, there are few alerts relating to it. But at a university there should be massive traffic of this kind. The copyright discussion might be of no relevance to the University, but the massive security problems with the p2p protocols should be (e.g. see William Couch's practical about these risks). Note that there is signs of eDonkey and Kazaa traffic in the Snort scan and OOS files analysed.

- There is a peculiar situation with outside traffic going past the NIDS sensor. I think the University should revise its routing or network topology. Alternatively the IDS configuration does not correctly reflect the network.

- The sensor who has produced these logs seems to have a rule set that is not good enough. Some of the rules give out massive information, while lots of things seem to be missing. There seems to be a big false positive problem, so the rule set should be thoroughly tuned. For more on this see the comments on the individual alerts. I think it would be better to base the rules on the Snort default rule set, adding custom rules as needed. These give a better situation concerning documentation and exchange of security information with the Snort community.

References

To Part 1 - but also used in Parts 2 and 3:

1. Jay Beale/ James C. Foster/Jeffrey Posluns: "Snort 2.0 Intrusion Detection"

2. www.snort.org

3. www.whitehats.com

4. www.securityfocus.com

5. www.cert.org

6. google.org

7. Snort man page and rule set of Snort 2.0.0

To Parts 2 and 3:

8. Stephen Northcutt, Mark Cooper, Matt Fearnow/Karen Frederick "Intrusion Signatures and Analysis"

9. Stephen Northcutt, Judy Novak "Network Intrusion Detection. An Analyst's Handbook"

10. W. Richard Stevens: "TCP/IP Illustrated vol. 1"

11. Larry Wall, Tom Christiansen, Randal L. Schwartz: "Programming Perl. Second Edition"

12. RFC 793 Transmission Control Program

13. RFC 1644 T/TCP -- TCP Extensions for Transactions. Functional Specification

14. T/TCP home page www.kohala.com/start/ttcp.html

15. Phrack Magazine Volume 8, Issue 53 July "T/TCP vulnerabilities"

16. Mark Stacey: " T/TCP -- TCP for Transactions", Linux Journal February 1, 2000 - http://www.linuxjournal.com

17. Knut Bjørnstad: "GIAC GCIA Version 3.3 Practical Detect" - posting to intrusions@incidents.org July 14, 2003

18. Andrew Rucker Jones: Re: GIAC GCIA Version 3.3 Practical Detect - posting to intrusions@incidents.org commenting Knut Bjørnstad July 17, 2003

19. Knut Bjørnstad: " Re: LOGS: GIAC GCIA Version 3.3 Practical Detect: 1" - posting to intrusions@incidents.org commenting Nicolas Cop July 16, 2003

20. Ethereal documentation www.ethereal.org

21. p0f documentation http://www.stearns.org/p0f/

22. Bente Petersen: "SANS Intrusion Detection FAQ: What is p0f and what does it do?" http://www.sans.org/resources/idfaq/p0f.php

23. Disco Fingerprint tool http://www.altmode.com/disco/

24. Ettercap http://ettercap.sourceforge.net

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

26. Knut Bjørnstad: "GIAC GCIA Version 3.3 Practical Detect #2"" - posting to intrusions@incidents.orgJuly 23, 2003

28. "Ports used by trojans (2002-10-15)" http://www.simovits.com/nyheter9902.html

29. "Backdoor.Skun" http://securityresponse.symantec.com/avcenter/venc/data/backdoor.skun.html

30. Fyodor: "The Art of Port Scanning" http://www.insecure.org/nmap/nmap_doc.html

31. "I-051: FreeBSD T/TCP Vulnerability" May 19,1998 http://ciac.llnl.gov/ciac/bulletins/i-051.shtml

32. "Intrusec Alert: 55808 Trojan Analysis" Last update July 16, 2003 http://www.intrusec.com/55808.html'

33. Stumbler Alert - X-Force Security Center http://www.iis.net/iisEn/delivery/xforce/alert/detail.jsp?oid=22441

34. Lancope Virus Alert http://www.lancope.com/news/Virus_Alert_Trojan.htm

35. Dennis Fisher: "Trojan Picks Up Steam, Baffles Experts" http://www.eweek.com/article2/0,3959,1130759,00.asp

36. Typot. F-Secure Virus Description http://www.f-secure.com/v-descs/typot.shtml

37. Joe Stewart: "Re: sdbot variant and port 55808 activity" posting to incidents@securityfocus.com Jun 20 2003

38. Todd A. Beardsley: "Intrusion Detection and Analysis: Theory, Techniques and Tools" GCIA Practical

39. William Couch GSEC Practical: "Peer-to-Peer File Sharing Networks: Security Risks"

40. Les M. Gordon: "Intrusion Analysis - The Directors Cut" GCIA Practical

41. Teri Bidwell GIAC GCIA Practical, October 2000

42. On the FreeBSD T/TCP configuration errors http://freebsd.ntu.edu.tw/squid/FAQ/FAQ-13.html

43. Donald Gregory: GIAC Intrusion Detection In-Depth GCIA 3.2 Practical

44. Datentausch-Dienste-Mini-FAQ
http://www.sockenseite.de/datentausch-minifaq.html.

Additional references

I did not want to change the reference numbers above so give these as an addition.

[45] Fyodor: "Nmap Remote OS Detection" http://www.insecure.org/nmap/nmap-fingerprinting-article.html

[46] http://www.uk.research.att.com/vnc/

[47] http://studentorganizations.smsu.edu/ ACM/Security/VNCSecurity.htm

[48] http://www.sans.org/resources/idfaq/index.php

[49] http://sourceforge.net/projects/ttcplinux/

[50] http://www.cert.org/current/archive/2003/07/31/archive.html#55808 on Stumbler

[51] http://www.irchelp.org Various supporting documentation, and FAQ on IRC

[52] IRC RFC's 1459, 2810, 2811, 2812, 2813.

To part 1:
[53] "An Overview of Issues in Testing Intrusion Detection Systems".
National Institute of Standards and Technology ITL: Peter Mell,
Vincent Hu, Massachusetts Institute of Technology Lincoln Laboratory: Richard
Lippmann, Josh Haines, Marc Zissman

[54] http://security.royans.net/info/posts/bugtraq_ddos3.shtml. Analysis of the shaft DDOS tool by Sven Dietrich

[55] http://security.royans.net/info/posts/bugtraq_ddos5.shtml. Analysis of the mstream DDOS tool by Dave Dittrich

[56] http://www.ntp.org/ntpfaq/NTP-s-config.htm#AEN2910. FAQ paragraph about NTP autentication.

[57] http://www.securityfocus.com/infocus/1205 Toby Millers explanation of the use of reserved bits in the TCP header, Queso and ECN