Global Information Assurance Certification Paper

GIAC
CERTIFIED INTRUSION ANALYST – GCIA
PRACTICAL ASSIGNMENT
VERSION 3.4

JOSÉ FAIAL
April 2004

Part 1 – Describe the State of Intrusion Detection

"Automatic Generation of Intrusion Signatures"

Abstract

Basically, all intrusion signatures are developed after an attack or vulnerability comes to public and sometimes this is too late. What if we can implement some way to catch new attacks automatically creating proper signatures that can dynamically feed our sensors? That's what *honeycomb* does. The way it's works, how to implement it and the results of some tests will be presented in this paper. It's expected from the reader, a basic understanding of Network Intrusion Detection technologies, as well how to interpret simple snort like signatures.

Preface

Modern Intrusion Detection Systems are changing the basic way they look for signs of malicious activities, from a simple pattern match to a complex mixture of protocol and behavior analysis, that points out only really interesting traffic, thus reducing the number of false positives and negatives. Those technologies are known as Protocol Analysis [1] and Anomaly (aka Behavior) Based Intrusion Detection [2]. They promise to improve IDS performance by reducing the number of pattern match signatures an IDS will have to deal with, also allowing the detection of known and unknown attacks, something impossible for common "grep like" signatures. The drawback with this approach is that many attacks don't rely on protocol violations or can't even be considered an anomaly, so intrusion detection based on pattern match signatures are going to still with us for a long time.

Static vs. Dynamic Based Intrusion Detection

Because of size constraints, I'll not go into the details of how a Protocol Analysis or an Anomaly Based Intrusion Detection engine works, instead is left for the interested readers some excellent references on these topics: [3][4][5][6][7][8]. After studying this material you'll understand that intrusion detection technologies can be splited in two categories: those based on static parameters and those based on dynamic decisions. Pattern match signatures are static because we need to both manually set some signature's field values as well install new ones periodically. A common pattern match signature extracted from Snort [9] database is show below:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-CGI phf access";
                        uricontent:"/phf"; nocase;)
```

Lot's of signature like that are needed to cover a range of known attacks and more signatures means more computer resources consumption. By the other side, dynamic detection engines comes with the required intelligence to learn what the expected behavior of the network is and to know what contents of a certain packet can have without requiring individual signatures. They are very good in detecting new buffer overflows attacks, portscans and covert channels [10]. Although, these technologies surfer from some problems. Actually, Anomaly Based engines are a very immature technology that needs a lot more research, they are difficult to implement and tuning. Protocol Analysis engines

lacks in supporting “not-so-common” protocols, i.e., they fail in detecting violations in a specific protocol a network may have. Another problem (that is not exactly a failure of the IDSes) is that many network device vendors does not strictly follows the protocol’s specifications, thus resulting in different implementations of the same protocol that the IDS engine will have to deal it. We must also consider that some attacks cannot be considered a protocol violation or even an anomaly, and what if we need to look for traffic of a custom made application? We must rely on common signatures, like the above example, to do it. And that’s exactly the greatest advantage of static signatures: they are flexible, easy to implement, tuning and if the detection engine uses a open language structure, to develop new ones. Combining these technologies may result in a powerful detection engine, and that’s what intrusion detection developers are doing: merging the better of all worlds.

Detecting new attacks although stills a hard task. Basically, all signatures are developed after an attack or vulnerability comes to public and sometimes this is too late. What if we can implement some way to catch new attacks automatically creating proper signatures that can dynamically feed our sensors? That’s what *honeycomb* [11] does. The way it’s works, how to implement it and the results of some tests will be presented in the next sections, but before we go deep into details of honeycomb, we need to open space for the introduction of a very interesting and exciting intrusion technology: *honeypots*. [12]

Honeypots

The better definition of a honeypot is extracted from a Lance Spitzner´s paper entitled “Open Source Honeypots: Learning with Honeyd” [13]:

> “*A honeypot is a security resource whose value lies in being probed, attacked, or compromised*”.

Because of above definition, all traffic directed to the honeypot is malicious by default. It makes sense because if no one knows about it and no official service are being provided by the honeypot, nobody should be talking to it. There is no point for real users to interact with these systems. There are two main types of honeypots called “high interaction” and “low interaction” that may be used for research or production purposes. A high interaction honeypot is a real system equipped with a working operating system, services and applications, while a low interaction is a virtually emulated system. Production honeypots are those used to protect a network, helping secure your organization, eluding attackers and freeing the real systems; research honeypots are used for information gathering with the purpose of implement an early warning system, discover new attacks techniques and exploits, counter-intelligence or law enforcement. For a complete in depth understanding of honeypots, please refer to the given references. We’ll focus on a fantastic low interaction honeypot called *honeyd* [14].

Honeyd

Honeyd is low interaction honeypot developed and maintained by Niels Provos. It’s a free Open Source system, which runs basically on unix-based operating systems, although it was ported to Windows. Its primary purpose is detection of unauthorized activities within an organization.

Honeyd works by monitoring all or some of unused IP addresses in the network. As soon a connection is attempted against one of those addresses, honeyd will assume the identity of the unused IP, simulating a real system and interacting with the attacker. Remember

that we should not expect to see any connections to inexistent systems in our networks. There is a high likelihood that it's in fact a probe, scan or a worm hitting the network. So, any time honeyd sees a connection and generates an alert, you know it's most likely a real attack and not a false one. To simulate a system or network of systems, honeyd makes use of a TCP/IP stack emulator based on NMAP's fingerprints database and some scripts to simulate the network services, like SMTP, HTTP and others. Those scripts are also called Fake Services. As an example, the piece of code below extracted from a honeyd script simulates a Sendmail SMTP server:

```
#!/bin/sh
# SMTP (Sendmail) Honeypot-Script intended for use with
# Honeyd from Niels Provos
# -> http://www.citi.umich.edu/u/provos/honeyd/
#
# Author: Maik Ellinger
# Last modified: 17/06/2002
# Version: 0.0.8

QUIT* )
          echo -e "220 2.0.0 $HOST.$DOMAIN closing connection\r"
           my_stop
                    ;;
RSET* )
          echo -e "250 2.0.0 Reset state\r"
                    ;;
HELP* )
          echo "214-2.0.0 This is sendmail version 8,12,2"
          echo "214-2.0.0 Topics:"
          echo "214-2.0.0       HELO    EHLO    MAIL    RCPT    DATA"
          echo "214-2.0.0       RSET    NOOP    QUIT    HELP    VRFY"
          echo "214-2.0.0       EXPN    VERB    ETRN    DSN     AUTH"
          echo "214-2.0.0       STARTTLS"
          echo "214-2.0.0 For more info use \"HELP <topic>\"."
          echo "214-2.0.0 To report bugs in the implementation send email to"
          echo "214-2.0.0       sendmail-bugs@sendmail.org."
          echo "214-2.0.0 For local information send email to Postmaster at your
site."
          echo "214 2.0.0 End of HELP info"
          ;;
```

As you can see from this part of the script, the commands "QUIT", "RSET" and "HELP" are being simulated like a real sedmail server. Many services and operating systems can be simulated by honeyd at same time. The honeyd´s configuration file is used to manage the virtual honeypot configuration. A sample configuration file that setups a Windows 2000 Advanced Server server is shown below:

"annotate "MS Windows2000 Professional RC1/W2K Advance Server Beta3" fragment old
create template
set template personality "MS Windows2000 Professional RC1/W2K Advance Server Beta3"
add template tcp port 80 "perl iisemul8.pl"
set template default tcp action reset
set template uid 32767 gid 32767
bind 10.1.1.1 template
set 10.1.1.1 uptime 1327650"

The statement "bind 10.1.1.1 template" tells honeyd to bind the virtual sever to 10.1.1.1 IP address and the "add template tcp port 80 "perl iisemul8.pl"" to start the "issemu8.pl" script when a connection is made to the HTTP port.

Another great characteristic of honeyd is that it logs all traffic detected against the honeypot, so we'll always have some sort of data to play with after the intrusion. This

flexibility makes honeyd a powerful tool in our security arsenal. Please, for a complete reference about honeyd, visit the Honeyd website at http://www.honeyd.org.

Now that we were introduced to the very basic concepts of honeypots and honeyd, we can go back to our discussion on automatic generation of intrusion signatures.

Honeycomb

Until now, all data captured by a honeypot must be manually analyzed by someone. Although this process stills crucial, it takes time to produce some useful results, such as new intrusion signatures, for example. Honeycomb helps reducing this delay, because it does it automatically. It creates a very powerful combination between honeypots and traditional intrusion detection sensors by turning our outdated-static-dumb pattern match engine into a very dynamic system.

Honeycomb is developed and maintained by Christian Kreibich of University of Cambridge's Computer Laboratory. His goal is to make honeycomb smart enough to automatically inspect the traffic inside the honeypot at different levels in the protocol hierarchy, producing signatures for malicious network traffic. Currently, it generates signatures for Bro [15] and Snort.

How it works

Let's see a little overview of how honeycomb works so we can understand how signatures are generated. We'll not enter into the internals of algorithms and all steps involved as this is fully explained by the honeycomb's creators in the paper "Honeycomb – Creating Intrusion Detection Signatures Using Honeypots" [11], instead we'll focus in a practical application of it.

Architecture

Honeycomb is an extension of honeyd. It "*runs as plugin that remains logically separated from the honeyd codebase, while an event hooks provide a mechanism to integrate it into the activities inside the honeypot"* [11]. The hooks allows a plugin to be always updated about the honeyd´s connection state, which data has been sent or received and which data is passed or received from the subsystems. The honeycomb's architecture is illustrated in Figure 1.

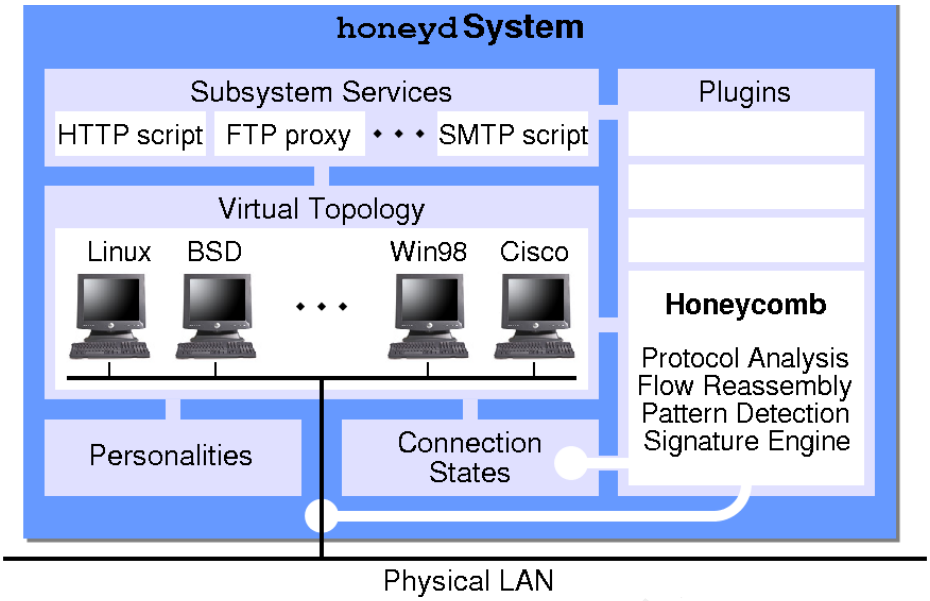


Figure 1: Architecture [16]

Signature Creation algorithm

For each packet that honeycombs intercepts, it initiates a similar sequence of activities. A briefly description of each phase is given below:

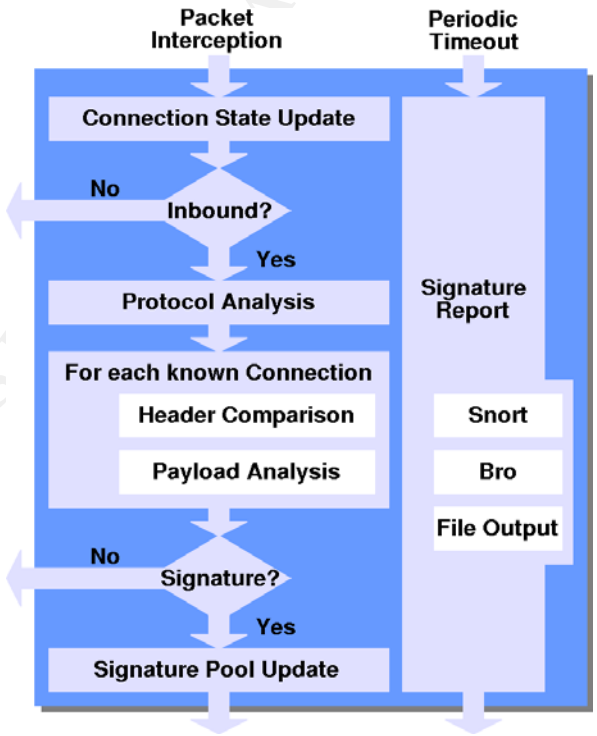


Figure 2: signature generation mechanism [16]

a) **Connection tracking**: maintains the state of UDP packets sent or received by the honeypot and TCP connections. As the goal is to generate signatures by comparing new traffic to previously seen ones, this information cannot be released

just after the connection is closed, instead the connection tracking mechanism builds state tables to keep it stored as long as needed. This allows the analysis of attacks that takes multiple steps to be accomplished.

b) **Protocol analysis**: deals with the analysis of some traffic characteristics, like address, ports, IP identifiers, sequence numbers and others. It does protocol analysis at network and transport layers for IP, TCP and UDP, spotting invalid fragmentation offsets, unusual TCP flags combinations and other anomalies.

c) **Pattern detection**: in this phase an algorithm called LCS is applied to the reassembled flow content. The role of LCS is to find payload patterns that can be used in the signature.

d) **Signature lifecycle**: at this point, if no interesting facts for a signature is found,, processing of currently packet ends and the analysis of a new one starts. Otherwise, a signature is added to a pool, which maintains a cache of recently detected signatures. Signatures are then compared to previously ones to check if they can be aggregated in order to generate an improved one, thus reducing the its number. The resulting signatures are reported periodically and the old signatures are removed from the cache when it becomes full.

e) **Signature output**: at this step, the contents of signature cache are sent to output modules that converts the records into Bro and Snort format. The signature strings are then, dumped to a file.

Playing with honeycomb

How to get it up and running

To get honeycomb up and running in your system, you need at least honeyd version 0.5c already installed in your system. To install honeycomb, follow these steps:

Download honeycomb and the required libstree library source code (available at http://www.cl.cam.ac.uk/~cpk25/downloads/libstree-0.4.0.tar.gz) and unpack them to a temporary directory. Assuming you are using the "/tmp" dir:

```
honeyprobe# cd /tmp
honeyprobe# tar xvzf libstree-0.4.0.tar.gz
honeyprobe# cd libstree-0.4.0
honeyprobe# ./configure & make & make install
honeyprobe# cd /tmp
honeyprobe# tar xvzf honeycomb-0.5.tar.gz
honeyprobe# cd honeycomb-0.5
honeyprobe# ./configure & make & make install
```

If everything goes well, we'll have honeycomb plugin available for use[1]. Next, we need to rebuild honeyd to make use of this new feature. Go to the directory where you placed the honeyd´s source files (I'll use /tmp/honeyd-0.8):

```
 honeyprobe# cd /tmp/honeyd-0.8
 honeyprobe# ./configure –with-plugins=honeycomb (plus any other plugin and
configuration option you may be using)
honeyprobe# make & make install
```

[1]The author had some problems when compiling honeycomb in his system related to syntax errors in "event.h" included C header file. This file is part of *eventlib*, a library required by honeyd as well and currently in version 0.8. The problem was fixed, downgrading libevent to version 0.7c and restarting the installation process.

This will build honeyd with honeycomb support. From now on, you may see the following message when starting honeyd if you start it by hand (also check your /var/log/messages file):

```
honeyd[“pid”]: registering plugin 'Honeycomb' (0.5)
```

Configuring honeycomb

Honeycomb’s configuration is done by using the honeyd´s config file. For helping us in the setup process, honeycomb package comes with a sample config file for using with honeyd. Look at honeyd.cnf file in the directory where you placed the honeycomb source and add its content to your honeyd configuration file. The default configuration should be fine for most setups, but you can tune it to fit your needs. Let’s take a look at some important fields:

option honeycomb enable 1
This is where we enable (1) or disable (0) honeycomb plugin. The default is enabled.

option honeycomb sig_output_file /tmp/honeycomb.log
This option sets the file where generated signatures will be published. The default is /tmp/honeycomb.log. Sets it to a more appropriated place, like /var/log/honeycomb (don’t forget to create it and set the proper permissions before running honeycomb)

option honeycomb udp_max_msg_size 5000
option honeycomb tcp_max_msg_size 5000
These are the maximum udp and tcp message string size the LCS algorithm will have to deal with. Setting it too high may degrade performance, so try to keep it as low as possible.

option honeycomb tcp_conns_max 65000
option honeycomb tcp_dataconns_max 1000
These options deals with the maximum number of initiated (still waiting for the end of three-way-handshaking) and established (three-way completed) ones, respectively. If you sit your honeypot in a very heavy network or have configured honeyd to simulate a large network topology, with dozens of virtual honeypots, you may have to increase these values. However, keep in mind that setting it too high may expose your system to a Denial of Service, because honeycomb can get too busy in case of a portscan.

option honeycomb tcp_max_buffering_in 1000
This defines the size of the buffer that stores incoming tcp payloads. You may have to increase this value if you plan to handle large payloads, like those founds in some buffer overflow attacks[2].

option honeycomb sighist_max_size 200
This is maximum number of signatures honeycomb will maintain before cleaning the old ones.

Now that we were introduced to some of honeycomb's options, lets play with it.

[2]To be able to catch a signature for the IIS Webdav exploit, the author had to increase its value. It was set to 5000 without experimenting significant degradation of his system.

Setup environment

Honeycomb was tested against two different scenarios. An isolated and controlled lab network and a real network connected to Internet. This was done so we could get familiar with it and their results before going to the real challenging Internet environment. The lab network was build using VMware Workstation 4.1 (http://www.vmware.com) to run the honeypot, while the host system was used to execute some attacks against the virtual network. Then, 0day exploits for recent vulnerabilities (at time of this writing) were launched. Let's see the results of executing the THCIISSLAME.c (http://www.thc.org/download.php?t=e&f=THCIISSLame.c) exploit against the honeypot. This exploits takes use of a flaw in the implementation of PCT protocol present in Microsoft's Internet Information Services web server and other SSL enabled products (more details available at http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0719) .

- Launching the exploit:

```
C:\tools>THCIISSLame.exe 192.168.1.125
THCIISSLame v0.1 - IIS 5.0 SSL remote root exploit
tested on Windows 2000 Server german/english SP4
by Johnny Cyberpunk (jcyberpunk@thc.org)
[*] building buffer
[*] connecting the target
[*] Exploit send successfully! Sleeping a while ....
[*] Trying to get a shell
can't connect to port 31337 ;( maybe firewalled ...
```

Note: as expected, the exploit "fails" because it's not being launched against a real vulnerable IIS host. For an unknown 0day exploits we never know what to expect, for example, this one tries to get a shell on port 31337 after sending the exploit. Our honeypot wasn't expecting for it, so we couldn't provide a fake shell for the attacker (the exploit appears to have failed). When running honeyd or any other honeypot, pay attention to all traffic that just follows the attack. It may give you some clues on what the attacker are expecting to see, so we can elude he got a successful victim next time.

- Signatures produced by honeycomb:

```
# Signature report at Sun May 2 21:52:00 2004

alert tcp 192.168.1.1/32 any -> 192.168.1.125/32 443 (msg: "Honeycomb Sun May  2
21h51m48 2004 "; flags: PA; flow: established; content: "|80|b|01 02 BD 00 01 00
01 00 16 8F 82 01 00 00 00 EB 0F|THCOWNZIIS!2^|BE 98 EB|#zi|02 05|lY|F8 1D 9C DE
8C D1|Lp|D4 03 F0|'  0|08|WS2_32.DLL|01 EB 05 E8 F9 FF FF FF|]|83 ED|*j0Yd|8B 01
8B|@|0C 8B|p|1C AD 8B|x|08 8D|_<|8B 1B 01 FB 8B|[x|01 FB 8B|K|1C 01 F9 8B|S$|01
FA|SQR|8B|[ |01 FB|1|C9|A1|C0 99 8B|4|8B 01 FE AC|1|C2 D1 E2 84 C0|u|F7 0F
B6|E|05 8D|DE|04|f9|10|u|E1|f1|10|ZX^VPR+N|10|A|0F B7 0C|J|8B 04 88 01 F8 0F
B6|M|05 89|D|8D D8 FE|M|05|u|BE FE|M|04|t!|FE|M"|8D|]|18|S|FF D0 89
C7|j|04|X|88|E|05 80|Ew|0A 8D|]t|80|k&|14 E9|x|FF FF FF 89 CE|1|DB|SSSSVFV|FF D0
97|UXf|89|0j|10|UW|FF|U|D4|NVW|FF|U|CC|SUW|FF|U|D0 97
8D|E|88|P|FF|U|E4|UU|FF|U|E8 8D|D|05 0C 94|Sh.exeh\cmd|94|1|D2 8D|E|CC
94|WWWSS|FE C6 01 F2|R|94 8D|ExP|8D|E|88|P|B1 08|SSj|10 FE CE|RSSSU|FF|U|EC|j|FF
FF|U"; )

alert tcp 192.168.1.1/32 any -> 192.168.1.125/32 443,31337 (msg: "Honeycomb Sun
May  2 21h51m50 2004 "; flags: S; flow: stateless; )
```

The first signature catches the exploit; the second shows that a connection to port tcp 31337 was requested (note the SYN flag). Let's compare honeycomb's output to honeyd´s log. The contents of honeyd.log file shows that a connection to port 443 and three sequentially connections to port 31337 were made (Can you guess why three connections were attempted? It's the standard number of Windows XP connection retries, also note the

characteristic delays between each retry):

```
2004-05-02-21:17:41.7732 tcp(6) S 192.168.1.1 3297 192.168.1.125 443 [Windows XP SP1]
2004-05-02-21:17:43.3368 tcp(6) - 192.168.1.1 3298 192.168.1.125 31337: 48 S [Windows XP
SP1]
2004-05-02-21:17:43.8094 tcp(6) - 192.168.1.1 3298 192.168.1.125 31337: 48 S [Windows XP
SP1]
2004-05-02-21:17:44.2990 tcp(6) - 192.168.1.1 3298 192.168.1.125 31337: 48 S [Windows XP
SP1]
```

and the web.log file shows the following:

```
--MARK--,"Sun May  2 21:51:48 EDT
2004","IIS/HTTP","192.168.1.1","192.168.1.125",3297,443,
"€b½
",
--ENDMARK--
```

It's not very useful data. One can note we are faking a HTTPS session using a HTTP emulator, and this is not going to work well, but this works fine against attacks that doesn't care with the server's responses. Comparing the above results, we can see that honeycomb is really a powerful complement to honeyd. However, a skilled intrusion analyst may say that the generated signature is not a "good signature", because it's too big, so it will take a lot of CPU and memory resources for the IDS sensor to process it. That's right for sure, but now that we **knows** the signature, we can tune it! Nobody will blame honeycomb for generating that "too-big-signature" that caught that last 0day worm and saved your skin! Note that **`"THCOWNZIIS"`** and **`"WS2_32.DLL"`** strings may be used to tune up the signature, i. e., making a small one. Now let's look at what honeycomb have found for us in a real honeypot setup:

> *"It's Friday April 30, 2004. The honeypot was installed in a segment of a huge enterprise network for purpose of testing and setup before going into production. For the first setup, honeyd was simulating only three Windows XP boxes running Internet Information Services web server. Common Windows TCP/UDP ports were open. I wasn't expecting to catch anything during the setup, so I left it "collecting" some data just to be familiar with how honeyd would behave in my network.*
>
> *Saturday, May 1. Sasser [17] worm starts spreading trough Internet. As everyone else on the field of network security, I give a hold on the weekend and went to the office to support the SOC guys in anything they need. For our luck, and thanks for the early updates, everything was quite normal. Same during Sunday, but we were expecting to see some action on Monday when "normal" people starts working after the weekend and most computers are turned on, so we placed some filters in the routers to block the ports used by the worm. After the third detected occurrence of the worm, I was very curious to see what the honeypot was seeing. Looking at honeyd logs I could detect some probes to port 445 that wouldn't be a surprise because many broadcasts are associated to Windows networking, and some other probes to port... 9996!? That's no good! Incident handling steps followed, infected machine cleaned and updated, I went back to my honeypot again. Honeycomb log shows some signatures... a quick analysis showed they are related to Windows networking (I need to have a serious talk with the Windows support guys!), nothing that snort don't already have... some lines below... bingo! What are these signatures about?*
>
> ***alert tcp 10.18.0.0/16 any -> 10.18.74.0/24 9996 (msg: "Honeycomb Mon May 3 6h43m58 2004 "; flags: PA; flow: established; content: "5554>>cmd.ftp&echo anonymous>>cmd.ftp&echo user&echo bin>>cmd.ftp&echo get";)***
>
> ***alert tcp 10.18.0.0/16 any -> 10.18.74.0/24 9996 (msg: "Honeycomb Mon May 3 6h44m04 2004 "; flags: PA+; flow: established; content: "5554>>cmd.ftp&echo anonymous>>cmd.ftp&echo user&echo bin>>cmd.ftp&echo get";)***
>
> *They are related to the sasser worm. Specifically, the worm has infected some machine and was*

sending commands trough a shell opened on port 9996. Those commands, instructs the infected machine to download the worm via ftp on port 5554 from the infecting machine. I did some modification in the signatures (they are pretty similar excluding the flag stuff, so I pick the one that I think would work better):

alert tcp any any -> $HOME_NET 9996 (msg: "Sasser.A FTP Commands Detected"; flags: PA*; flow: established; content: "5554>>cmd.ftp&echo anonymous>>cmd.ftp&echo user&echo bin>>cmd.ftp&echo get"; flow:to_server,established; resp:rst_all; priority:1; sid: 1000005;)

We published this signature to our sensors around the network and were able to catch a dozen machines infected. Thanks to honeyd and honeycomb!

Ok, I confess we already had a signature in place to detect scans for port 9996, but this one is so cool I used it to prioritize alerts and reset connections of machines that are getting trough locations without filters and gaining a successful shell. "

That's a true history! All characters used in this history are real :-)

Conclusion

I have been playing with honeycomb for a while now, but I still learning how to use it better. I find interesting results everyday, but I think some improvements are needed, like the possibility to define "bpf like" filters to ignore the analysis of some traffic and a "black list" of signatures we don't want to be generated again. I found it a very powerful and useful tool. It really aggregates a new functionality to honeyd I suggest to everyone have.

The output signatures need some analysis and minor modifications, like a new name, to avoid having duplicated or unneeded ones, so an intrusion analyst is necessary to use it well. I learned honeycomb is not a tool for beginners, despite its ease setup and configuration. It really automates the generation of new signatures, but at its current version, most need some kind of tuning before going into production. The developers are working to improve the output plugins to take advantage of more modern snort's signature statements, thus resulting in better signature performance. I can't wait to put my hands in the upcoming versions!

References

[1] Tanase, Matt. "The Great IDS Debate: Signature Analysis Versus Protocol Analysis", 2003, SecurityFocus, http://www.securityfocus.com/infocus/1663

[2] Liston, Kevin. "Can you explain traffic analysis and anomaly detection?" in SANS Intrusion Detection FAQ, http://www.sans.org/resources/idfaq/anomaly_detection.php

[3] "Del" Elson, David. "Intrusion Detection, Theory and Practice", SecurityFocus, 2003, http://www.securityfocus.com/infocus/1203

[4] Allen, J., Christie, A., Fithen, W., McHugh, J., Pickel, J., and Stoner, E., State of the Practice of Intrusion Detection Technologies. CMU/SEI-99-TR-028, Carnegie Mellon University, Software Engineering Institute, January 2000.

[5] Debar, Herve. "What is knowledge-based intrusion detection?" in SANS Intrusion Detection FAQ, http://www.sans.org/resources/idfaq/knowledge_based.php

[6] Debar, Herve. "What is behavior-based intrusion detection?" in SANS Intrusion Detection FAQ, http://www.sans.org/resources/idfaq/behavior_based.php

[7] Denning, Dorothy, "An Intrusion-Detection Model", IEEE Symposium on Security and Privacy, 1986, http://www.cs.georgetown.edu/~denning/infosec/ids-model.rtf

[8] Tanase, Matt, “One of These Things is not Like the Others: The State of Anomaly Detection”, SecurityFocus, 2003, http://www.securityfocus.com/infocus/1600

[9] M. Roesch, “Snort: Lightweight Intrusion Detection for Networks,” in Proceedings of the 13th Conference on Systems Administration, 1999, http://www.snort.org/docs/lisapaper.txt

[10] Ruiz, John. “Understanding covert channels,” GIAC GCIA Practical Assignment, 2003, http://www.giac.org/practical/GSEC/Jim_Goltz_GSEC.pdf

[11] Kreibich, Christian; Crowcroft, Jon, “Honeycomb - Creating Intrusion Detection Signatures Using Honeypots,” in HotNets-II Talks, 2003, http://www.cl.cam.ac.uk/~cpk25/publications/honeycomb-hotnetsII.pdf

[12] Spitzner, Lance, “Honeypots: Definitions and Value of Honeypots,” tracking-hackers.com, 2003, http://www.tracking-hackers.com/papers/honeypots.html

[13] Spitzner, Lance, “Open Source Honeypots: Learning with Honeyd,” SecurityFocus, 2003, http://www.securityfocus.com/infocus/1659

[14] Provos, Niels, “Honeyd - A Virtual Honeypot Daemon”, in 10th DFN-CERT Workshop, 2003, http://www.citi.umich.edu/u/provos/papers/honeyd-eabstract.pdf

[15] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time,” Computer Networks, 1998, http://citeseer.nj.nec.com/article/paxson98bro.html

[16] Kreibich, Christian; Crowcroft, Jon, “Honeycomb - Automated IDS Signature Generation using Honeypots,” in HotNets-II Talks, 2003, http://www.cl.cam.ac.uk/~cpk25/talks/2003-hotnets-honeycomb/honeycomb.ppt

[17] LURHQ Threat Intelligence Group, “Analysis of Sasser worm,” 2004, http://www.lurhq.com/sasser.html

Part II – Network Detects

Network Detect 1 – DOS BGP spoofed connection reset attempt

```
[**] [1:2523:2] DOS BGP spoofed connection reset attempt [**]
[Classification: Attempted Denial of Service] [Priority: 2]
11/18-19:08:20.235607 0:4:76:45:61:39 -> 0:50:56:40:0:6D type:0x800 len:0x3E
10.10.10.195:2844 -> 172.20.11.80:179 TCP TTL:128 TOS:0x0 ID:38868 IpLen:20 DgmLen:48 DF
******S* Seq: 0x3254E63B  Ack: 0x0  Win: 0xFAF0  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
[Xref => http://www.uniras.gov.uk/vuls/2004/236929/index.htm][Xref =>
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0230]

--

[**] [1:2523:2] DOS BGP spoofed connection reset attempt [**]
[Classification: Attempted Denial of Service] [Priority: 2]
11/18-19:08:20.311264 0:50:56:40:0:6D -> 0:4:76:45:61:39 type:0x800 len:0x3C
172.20.11.80:179 -> 10.10.10.195:2844 TCP TTL:62 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0  Ack: 0x3254E63C  Win: 0x0  TcpLen: 20
[Xref => http://www.uniras.gov.uk/vuls/2004/236929/index.htm][Xref =>
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0230]
```

Explanation of the Snort Alert Log:

11/18-19:08:20.311264	-	date and time of the alert
0:50:56:40:0:6D	-	source Mac Address
0:4:76:45:61:39	-	destination mac Address
type:0x800	-	encapsulating protocol
len:0x3C	-	length of frame 0x03c = 60
172.20.11.80	-	source address
179	-	source port
10.10.10.195	-	destination address
2844	-	destination port
TCP	-	Protocol type
TTL:62	-	Packet TTL
ID:0	-	IP ID
IpLen:20	-	Length of IP packet (bytes)
DgmLen:40	-	length of entire datagram inc head and payload (bytes)
DF	-	Do Not Fragment Flag
***A*R**	-	TCP Flags set (ACK and RST)
Seq: 0x0	-	TCP Sequence Number
Ack: 0x0	-	TCP Acknowledgment number
Win: 0x0	-	TCP Window Size
TcpLen: 20	-	Length opf TCP packet header (bytes)
[Xref =>]	-	any cross references for the alert

Source of Trace:

The trace was taken from raw log files downloaded from Internet Storm Center's web site at URL http://www.incidents.org/logs/RAW/2000.12.15.tgz . The tarball contains 14 consecutive days of data generated by a snort instance logging in binary mode. Information contained in that site explains that true network addresses were obfuscated and checksums modified to avoid computing the original IP addresses from it, but I failed to find one single packet with bad checksums, indicating that IPs were obfuscated using a technique other than doing an simple "find-and-replace" or even no obfuscation have been done at all. All of 14 files were merged on one single file for processing with snort as follow:

```
[root@white logs] mergecap -w allcaps.cap 2003.12.15.*
```

Many commands and procedures used in this section were learned and influenced from posted GCIA practicals works of Peter Storm, Les Gordon and Ian Martin (thanks guys for the great work). Those documents can be found at GIAC Web Site's posted practical section at http://www.giac.org .

I'll try to figure out the network topology doing an analysis in the MACs found. A deep analysis in the MAC address found in this file shows a dozens of different network cards. This indicates that the sensor is located between the internal network and the gateway or the sensor may be placed in a SPAN port of the LAN switch that is mirroring the trunking port or a VLAN, but I'll present other possibility based on my findings later. I'll try to figure out which MACs belongs to local computers and which ones to gateways counting how many IP addresses are associated to each MAC. First, what are the source MAC addresses present in the captured file and how many packets are associated to each one:

```
tcpdump -ner 2003.12.15.7 | awk '{print $2}'| sort | uniq -c | sort
      1 0:9:6b:2:e9:3d
      5 0:c:29:14:1e:63
     11 0:0:e2:92:ee:f
     13 0:0:e2:94:b0:2a
     34 0:50:56:40:0:64
     35 0:b:db:17:f4:c9
     80 0:d:bc:17:4:ce
     80 0:d:bc:17:4:cf
     80 0:d:bc:17:4:d0
     80 0:d:bc:17:4:d2
     80 0:d:bc:17:4:d4
     80 0:d:bc:17:4:d5
     80 0:d:bc:17:4:d6
     80 0:d:bc:17:4:d8
    130 0:a:95:d9:95:84
    135 0:2:a5:b6:e2:e3
    146 0:e0:98:a1:7f:da
    151 0:1:3:88:29:92
    183 0:3:ff:df:95:84
    261 0:a0:c9:ba:6d:85
    463 0:8:74:5:b7:f8
   1015 0:3:47:8c:89:c2
   2261 0:c:29:9e:ef:53
   2344 0:d0:59:c6:5e:14
   3880 0:1:2:79:91:ed
  10861 0:4:76:45:61:39
  14230 0:50:56:40:0:6d
```

What are the destinations MACs?

```
tcpdump -ner 2003.12.15.7 | awk '{print $3}'| sort | uniq -c | sort
      2 0:b:db:17:f4:c9
      3 1:0:5e:0:0:2
      3 1:0:5e:0:0:5
      3 1:0:5e:0:0:6
      3 1:0:5e:7a:a:8c
      5 0:c:29:14:1e:63
      8 0:0:e2:92:ee:f
     12 0:d:bc:17:4:ce
     12 0:d:bc:17:4:cf
     12 0:d:bc:17:4:d0
     12 0:d:bc:17:4:d2
     12 0:d:bc:17:4:d4
     12 0:d:bc:17:4:d5
     12 0:d:bc:17:4:d6
     12 0:d:bc:17:4:d8
     13 0:0:e2:94:b0:2a
     16 0:1:2:79:91:ed
```

```
     29 0:50:56:40:0:64
     32 1:0:c:0:0:0
     48 1:0:c:cc:cc:cc
     49 ff:ff:ff:ff:ff:ff
     76 0:a:95:d9:95:84
     93 0:e0:98:a1:7f:da
     99 0:2:a5:b6:e2:e3
    127 0:3:ff:df:95:84
    132 0:1:3:88:29:92
    200 0:a0:c9:ba:6d:85
    239 0:c:29:9e:ef:53
    346 0:8:74:5:b7:f8
    464 1:80:c2:0:0:0
    646 0:3:47:8c:89:c2
   1454 0:d0:59:c6:5e:14
  10804 0:4:76:45:61:39
  21809 0:50:56:40:0:6d
```

Let's find which ones belong to gateways looking how many IP addresses are associated to each one. The following command was used to find this information:

tcpdump -ner 2003.12.15.7 ether src [SOURCE MAC] | awk '{print $6}'| awk -F \. '{print $1 "." $2 "." $3 "." $4}' | sort -u

The ones that have more than one IP associated with are:

0:3:47:8c:89:c2 – 10.10.10.165, 192.168.117.1, 192.168.213.1
0:50:56:40:0:6d – 10.10.10.1 10.30.30.2 172.20.11.1 172.20.11.2 172.20.11.52 172.20.11.80 172.20.201.1 172.20.201.135 172.20.201.198 172.20.201.2 192.168.17.135 192.168.17.2 192.168.17.66 192.168.17.68

0:b:db:17:f4:c9 – 0.0.0.0 10.10.10.194 169.254.135.50 (think this one is related to Windows 2000 autoconfiguration and can be ignored)

0:d0:59:c6:5e:14 – 10.10.10.141 238.122.10.140 (multicast, ignore it)

So, it appears that we have two gateways **0:3:47:8c:89:c2** and **0:50:56:40:0:6d,** however packets arriving from 0:3:47:8c:89:c2 have identical TTLs (TTL 128) and sequential IP IDs indicating that 192.168.117.1 and 192.168.213.1 are certainly spoofed:

15:09:13.500109 0:3:47:8c:89:c2 0:50:56:40:0:6d 0800 92: 192.168.117.1.137 > 172.20.201.1.137: [udp sum ok](ttl 128, id **43661**, len 78)
15:09:13.500161 0:3:47:8c:89:c2 0:50:56:40:0:6d 0800 92: 192.168.213.1.137 > 172.20.201.1.137: [udp sum ok](ttl 128, id **43662**, len 78)
15:09:13.688565 0:3:47:8c:89:c2 0:50:56:40:0:6d 0800 62: 10.10.10.165.1881 > 192.168.22.207.1080: S [tcp sum ok] 723317621:723317621(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 128, id **43663**, len 48)
<snip>
15:09:15.000774 0:3:47:8c:89:c2 0:50:56:40:0:6d 0800 92: 192.168.117.1.137 > 172.20.201.1.137: [udp sum ok](ttl 128, id **43701**, len 78)
15:09:15.000778 0:3:47:8c:89:c2 0:50:56:40:0:6d 0800 92: 192.168.213.1.137 > 172.20.201.1.137: [udp sum ok](ttl 128, id **43702**, len 78)
15:09:15.018746 0:3:47:8c:89:c2 0:50:56:40:0:6d 0800 62: 10.10.10.165.1897 > 192.168.17.66.80: S [tcp sum ok] 725076644:725076644(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 128, id **43703**, len 48)

Explanation of key TCPDUMP´s fields:

[**15:09:15.018746**][1] [**0:3:47:8c:89:c2**][2] [**0:50:56:40:0:6d**][3] 0800 62: [**10.10.10.165.1897**][4] [**>**][5] [**192.168.17.66.80**][6] : [**S**][7] [tcp sum ok] [**725076644:725076644**][8] [(**0**)][9] [**win 16384**][10] [<**mss 1460,nop,nop,sackOK**>][11] [(**DF**)][12] [(**ttl 128, id 43703, len 48**)][13]

1 date

2	source MAC address
3	destination Mac address
4	source IP address. source port
5	> direction of packet
6	destination IP address. destination port
7	TCP Flags
8	SEQ Number/Offset
9	Number of bytes in IP payload (ie TCP data)
10	TCP Window size
11	TCP Options
12	Fragment/Don´t Fragment Flag
13	TTL, IP ID and Length

Which hosts are leaving the network trought 0:50:56:40:0:6d?

tcpdump -ner 2003.12.15.7 ether dst 0:50:56:40:0:6d | awk '{print $6}' | awk -F \. '{print $1 "." $2 "." $3 "." $4}' | sort -u

10.10.10.112 / 10.10.10.141 / 10.10.10.147 / 10.10.10.165 / 10.10.10.174 / 10.10.10.186 / 10.10.10.195 / 10.10.10.196 / 10.10.10.222 / 10.10.10.224 / 10.10.10.226 / 10.10.10.228 / 10.10.10.232 / 10.10.10.234

Where they are going to?

tcpdump -ner 2003.12.15.7 ether dst 0:50:56:40:0:6d | awk '{print $8}'| awk -F \. '{print $1 "." $2 "." $3 "." $4}' | sort -u

10.10.10.1 / 149.134.52.149 / 172.20.11.1 / 172.20.11.2 / 172.20.11.52 / 172.20.11.80 / 172.20.201.1 / 172.20.201.135 / 172.20.201.198 / 172.20.201.2 / 172.20.201.3 / 192.168.17.1 / 192.168.17.135 / 192.168.17.66 / 192.168.17.67 / 192.168.17.68 / 192.168.22.207 / 198.123.30.132 / 198.41.0.5

Which hosts are coming from 0:50:56:40:0:6d ?

tcpdump -ner 2003.12.15.7 ether src 0:50:56:40:0:6d | awk '{print $6}'| awk -F \. '{print $1 "." $2 "." $3 "." $4}' | sort -u

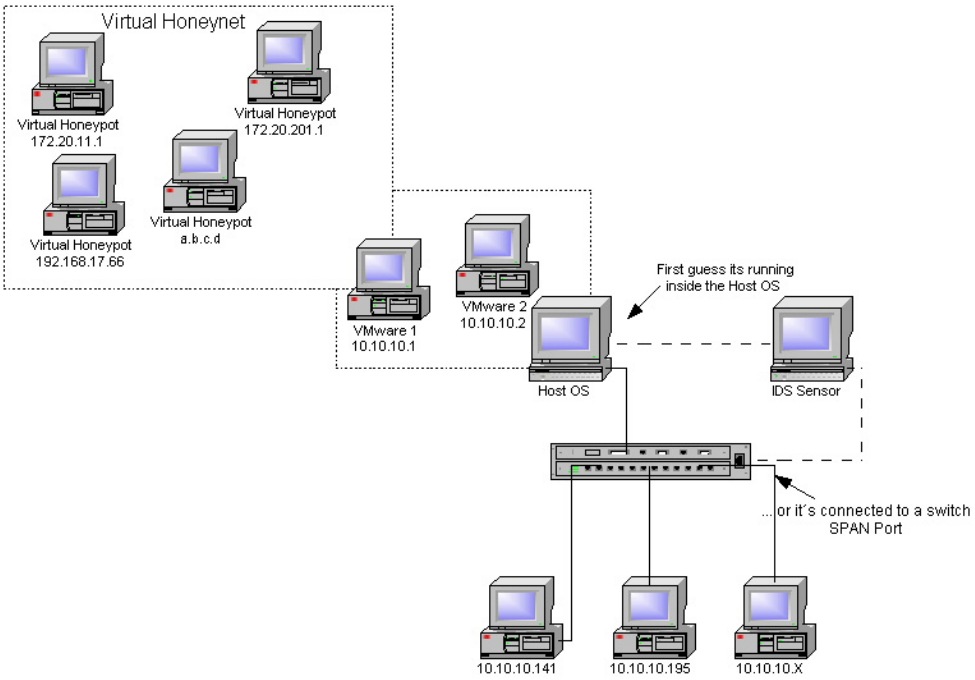
10.10.10.1 / 10.30.30.2 / 172.20.11.1 / 172.20.11.2 / 172.20.11.52 / 172.20.11.80 / 172.20.201.1 / 172.20.201.135 / 172.20.201.198 / 172.20.201.2 / 192.168.17.135 / 192.168.17.2 / 192.168.17.66 / 192.168.17.68

Where they are going to?

tcpdump -ner 2003.12.15.7 ether src 0:50:56:40:0:6d | awk '{print $8}'| awk -F \. '{print $1 "." $2 "." $3 "." $4}' | sort -u

10.10.10.1 / 10.10.10.112 / 10.10.10.141 / 10.10.10.147 / 10.10.10.165 / 10.10.10.174 / 10.10.10.186 / 10.10.10.195 / 10.10.10.196 / 10.10.10.222 / 10.10.10.224 / 10.10.10.226 / 10.10.10.228 / 10.10.10.232 / 10.10.10.234

It's interesting to note that **0:50:56:40:0:6d** and **0:50:56:40:00:64** (that is a DNS and DHCP server) are VMware cards. This information was taken from Ethereal´s ***manuf*** file and confirmed at IEEE database located at following web site address: http://standards.ieee.org/regauth/oui/oui.txt. This makes me speculate which kind of gateway and DNS server would be using a VMware card. My best guess is that 0:50:56:40:0:6d is a honeypot that runs honeyd and simulates hosts on the 10.30.30.x, 172.20.11.x, 172.20.201.x and 192.168.17.x networks. The honeypot has an IP address in the local subnet, which is 10.10.10.1, that connects it directly to the guest VMware OS that runs honeyd. My second guess is that the same host is hosting another VMware virtual host on 10.10.10.2 that runs DHCP and DNS services. My analysis results in a network topology similar to the figure below:



As a final note, it appears now that data was captured by a sniffer probe like tcpdump instead (or snort running in packet capture mode only). The capture file has a lot of packets related to Cisco protocols like CDP (Cisco Discovery Protocol), VTP and Spanning Tree that points out some useful information like the switch IP address (192.168.1.2), type of device, interfaces and some of its features (performs layer 3 routing, transparent bridging etc).

Detect was generated by:

The detect was generated by running Snort Version 2.1.2 (Build 25) using full ruleset base as available on May 05, 2004. The following command line switches were used:

```
# snort -c /etc/snort/snort.conf -r allcaps.cap -NUX -k none -de -l
/var/log/snort
```

-c: tells snort to read the configuration file /etc/snort/snort.conf
-r: tells snort to read data from a captured data file instead of sniffing it directly from the wire. For instance, the file *allcaps.cap* has been used
-N: disables logging. Alerts still working
-U: use UTC for timestamps
-X: dump the raw packet data starting at the link layer
-k: checksum mode to be used (all,noip,notcp,noudp,noicmp,none). For instance, checksums were ignored
-d: dump the Application Layer
-e: display the second layer header info
-l: tell snort to logo to directory /var/log/snort

The resulting output of snort processing was:

```
===============================================================

Snort processed 475199 packets.
Breakdown by protocol:                Action Stats:
```

```
    TCP: 372578     (78.405%)         ALERTS: 29281
    UDP: 66543      (14.003%)         LOGGED: 29281
   ICMP: 9986       (2.101%)          PASSED: 0
    ARP: 1329       (0.280%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 23582      (4.963%)
===============================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0          (0.000%)
    Control Packets:    0          (0.000%)
    Data Packets:       0          (0.000%)
===============================================================
Fragmentation Stats:
Fragmented IP Packets: 6          (0.001%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
===============================================================

TCP Stream Reassembly Stats:
   TCP Packets Used:      0          (0.000%)
   Reconstructed Packets: 0          (0.000%)
   Streams Reconstructed: 0
===============================================================
```

The below traffic:

```
15:08:16.623966 10.10.10.195.2551 > 172.20.11.80.179: S [tcp sum ok] 828765207:828765207(0) win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 38115, len 48)
15:08:16.659951 172.20.11.80.179 > 10.10.10.195.2551: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 62, id 0, len 40)
15:08:19.518648 10.10.10.195.2834 > 172.20.11.80.179: S [tcp sum ok] 843877911:843877911(0) win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 38689, len 48)
15:08:19.525274 172.20.11.80.179 > 10.10.10.195.2834: R [tcp sum ok] 0:0(0) ack 843877912 win 0 (DF) (ttl 62, id 0,
len 40)
15:08:19.787290 10.10.10.195.2844 > 172.20.11.80.179: S [tcp sum ok] 844424763:844424763(0) win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 38764, len 48)
15:08:19.813071 172.20.11.80.179 > 10.10.10.195.2844: R [tcp sum ok] 0:0(0) ack 844424764 win 0 (DF) (ttl 62, id 0,
len 40)
15:08:19.934638 10.10.10.195.2834 > 172.20.11.80.179: S [tcp sum ok] 843877911:843877911(0) win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 38781, len 48)
15:08:19.945200 172.20.11.80.179 > 10.10.10.195.2834: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 62, id 0, len 40)
15:08:20.235607 10.10.10.195.2844 > 172.20.11.80.179: S [tcp sum ok] 844424763:844424763(0) win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 38868, len 48)
15:08:20.311264 172.20.11.80.179 > 10.10.10.195.2844: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 62, id 0, len 40)
15:08:20.436254 10.10.10.195.2834 > 172.20.11.80.179: S [tcp sum ok] 843877911:843877911(0) win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 38917, len 48)
15:08:20.470553 172.20.11.80.179 > 10.10.10.195.2834: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 62, id 0, len 40)
15:08:20.737248 10.10.10.195.2844 > 172.20.11.80.179: S [tcp sum ok] 844424763:844424763(0) win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 38954, len 48)
15:08:20.788205 172.20.11.80.179 > 10.10.10.195.2844: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 62, id 0, len 40)

<snip>...

15:15:09.223696 172.20.11.80.179 > 10.10.10.195.1604: R [tcp sum ok] 0:0(0) ack 1271144247 win 0 (DF) (ttl 62, id 0,
len 40)
15:15:09.356124 10.10.10.195.1593 > 172.20.11.80.179: S [tcp sum ok] 1270559789:1270559789(0) win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 58770, len 48)
15:15:09.397537 172.20.11.80.179 > 10.10.10.195.1593: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 62, id 0, len 40)
15:15:09.657037 10.10.10.195.1604 > 172.20.11.80.179: S [tcp sum ok] 1271144246:1271144246(0) win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 58801, len 48)
15:15:09.677635 172.20.11.80.179 > 10.10.10.195.1604: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 62, id 0, len 40)
15:15:09.857749 10.10.10.195.1593 > 172.20.11.80.179: S [tcp sum ok] 1270559789:1270559789(0) win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 58821, len 48)
15:15:09.895531 172.20.11.80.179 > 10.10.10.195.1593: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 62, id 0, len 40)
15:15:10.158650 10.10.10.195.1604 > 172.20.11.80.179: S [tcp sum ok] 1271144246:1271144246(0) win 64240 <mss
```

1460,nop,nop,sackOK> (DF) (ttl 128, id 58856, len 48)
15:15:10.173672 172.20.11.80.179 > 10.10.10.195.1604: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 62, id 0, len 40)

Has triggered the signature…

alert tcp $EXTERNAL_NET any <> $HOME_NET 179 (msg:"DOS BGP spoofed connection reset attempt"; flow:established; flags:RSF; threshold:type both,track by_dst,count 10,seconds 10; reference:cve,CAN-2004-0230; reference:url,www.uniras.gov.uk/vuls/2004/236929/index.htm; classtype:attempted-dos; sid:2523; rev:2;)*

…and generated the reported alerts.

The signature above has the following meaning:

Send an alert if:
- TCP traffic flowing to or from $HOME_NET on port 179
  alert tcp $EXTERNAL_NET any <> $HOME_NET 179
- is part of an established connection
  flow:established
- has one of the RESET, SYN or FIN TCP flags set
  flags:RSF*
- occurs 10 times in a interval of 10 seconds for a same destination IP
  threshold:type both,track by_dst,count 10,seconds 10

Probability Source address was spoofed:

High. Spoofing a legitimate IP address that makes part of a BGP routing updating session is the basic element of this attack. Although, later analysis shows this detect is in fact a false positive, I'll keep this analysis as is for didactical purposes.

Description of Attack:

This attack utilizes a design flaw on RFC 793 and RFC 1323 TCP Extensions for High Performance, that allows an established TCP connection to be broken down by a third party attacker. The attacker's intention is to cause a Denial of Service condition on the network or user by sending especially crafted RST or SYN packets to the victims.

Gratuitous TCP resets as a way to disable a third party's connections is not a new thing. In fact, this is well known for long past, but it wasn't considered a high threat until a recent paper on this subject was published by Paul A. Watson. Originally, it was believed that a successful denial of service attack of this kind was not achievable in practice. The reason for this is that "*the receiving TCP implementation checks the sequence number of the RST or SYN packet, which is a 32 bit number, giving a probability of* $1/2^{32}$ *of guessing the sequence number correctly (assuming a random distribution*)" [quoted from http://*www.uniras.gov.uk/vuls/2004/236929/index.htm*]. The researcher have found that probability of guessing the acceptable sequence number is much higher than $1/2^{32}$ because the receiving TCP stack will accept any sequence number in a certain range of expected sequence number (that range in also called window). Higher the window, higher the probability. In fact, it's also possible to perform the attack using SYN packets, because the connection will be dropped if one of sides of a connection receives a duplicate SYN packet that has the ISN within the range of a previously established TCP connection window. This vulnerability relies in following statements of RFC 793:

> "*In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields [sequence numbers]. A reset*

is valid if its sequence number is in the window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN".

And

"*The principle reason for the three-way handshake is to prevent old duplicate connection initiations from causing confusion. To deal with this, a special control message, reset, has been devised. […] If the TCP is in one of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it aborts the connection and informs its user*".

Specifically, when done over TCP port 179 troughs an unprotected network device that supports and utilize BGP (Border Gateway Protocol), aka border routers, the attacker may cause a general failure in the routing infrastructure by suppressing routing updates and flapping. This can result in medium term network unavailability. BGP is potentially susceptible to this attack because:

a) It relies on persistent BGP connection between peers;

b) Source and destination IP addresses as well port number are very predictable (peers can be found using a traceroute for example)

Other application level protocols that are potentially affected have the following characteristics (extracted from http://*www.uniras.gov.uk/vuls/2004/236929/index.htm*):

a) *Depend on long lived TCP connections*

b) *Have known or easy-to-guess IP address end points*

c) *Have easy-to-guess source TCP ports*

CVE has created a candidate record for inclusion in the CVE List: CAN-2004-0230. Extracted from its description:

"*When using a large TCP Window Size, makes it easier for remote attackers to guess sequence numbers and cause a denial of service (connection loss) to persistent TCP connections by repeatedly injecting a TCP RST packet, especially in protocols that use long-lived connections, such as BGP*".

Attack Mechanism:

After the publication of Paul Watson´s paper, various tools to exploit this vulnerability were released. Most of then can be found at PacketStorm Security web site (http://www.packetstormsecurity.nl) searching for "TCP Reset" in the exploit section. Some of tools will compile on both Windows and Linux, there are even Perl versions of the exploit. To better understand how this exploit works and then compare what we learned against the captured traces, let's briefly analyze the traffic generated by one of available tools, AFX TCP Reset. A test lab was set up for purposing of testing the tool. It was a simple home made LAN consisting of three computers, two of them are peers in a lived TCP connection and the last one the attacker. Unfortunately, I don't have the available resources needed to run a test against a real BGP session, so I'll use simple "back-to-back" NetCat session. Lab environment:

Computer A: Server listening on 192.168.1.128 TCP port 53 (let's try to simulate a large DNS Zone Transfer)

Computer B: Client connecting from 192.168.1.134 source port 1026

Computer C: attacker (and windump probe) running on 192.168.1.1

Tool usage:
C:\tools>reset.exe
AFX TCP Reset
http://www.iamaphex.cjb.net
unremote@knology.net

Usage: reset <src ip> <src port> <dest ip> <dest port> <window size> <send delay> [begin seq num]

First opened NetCat shell on server:

```
[root@red-one root]#nc -l -p 53 -vv -n
Listening on [any] 53 ...
```

Connecting from client and watching the traffic:

```
[root@red-two root]#nc -p 1026 -vv -n 192.168.1.128 53
(UNKNOWN) [192.168.1.128] 53 (?) open
```

Windump out put of *3-way-handshake*:

C:\tools>windump -r c:\temp\reset.dump -vv tcp[13]!=0x14 (reset.dump stores the packets captured during this session)

02:00:03.197565 IP (tos 0x0, ttl 64, id 23154, len 60) 192.168.1.134.1026 > 192.168.1.128.53: S [tcp sum ok] **2144602788:2144602788**(0) win 5840 <mss 1460,sackOK,timestamp 282838 0,nop,wscale 0> (DF)

02:00:03.198499 IP (tos 0x0, ttl 64, id 0, len 60) 192.168.1.128.53 > 192.168.1.134.1026: S [tcp sum ok] **1010057679:1010057679**(0) ack **2144602789** win **5792** <mss 1460,sackOK,timestamp 4303997 282838,nop,wscale 0> (DF)

02:00:03.198664 IP (tos 0x0, ttl 64, id 23155, len 52) 192.168.1.134.1026 > 192.168.1.128.53: . [tcp sum ok] 1:1(0) ack **1** win 5840 <nop,nop,timestamp 282838 4303997> (DF)

Launching the attack:

C:\tools>reset 192.168.1.134 1026 192.168.1.128 53 5792 0 (have managed to not chose a initial sequence number)

After a few seconds, the server side of the connection ends abruptly, although nothing happened to the client side (this was expected as no connection state mechanism was being used by this dumb session). So, if no other packet was exchanged between peers during the attack period, a reset would occur in some place between 2144602790 and 2144608582 (because the window size of 5792). Let's try to find any packets that sit in this range. I opened the *reset.dump* file using Ethereal and applied the filter "*tcp.seq >= 2144602790 and tcp.seq <= 2144608582*" to the captured data, then exported the results to another file (*reset-packet.dump*) for processing with windump:

C:\tools>windump -r c:\temp\reset-packet.dump -vv -n
02:02:34.887431 IP (tos 0x0, ttl 128, id 12160, len 40) 192.168.1.134.1026 > 192.168.1.128.53: R [tcp sum ok] 2144604669:2144604669(0) ack 2144610461 win 40982

It shows only one packet found. That's the packet that shutdown my connection. Note that it was taken about 2 minutes and a half to close the connection with SEQ numbers starting at 249746077 (too far from real ISN – see the first RST packet captured below) and using a relatively small window. For a matter of curiosity, about 300K packets were generated before shutting down the session. Below is the first RST packet sent by the attacker:

C:\tools>windump -r c:\temp\reset.dump -vv -c 1 tcp[13]=0x14

02:00:27.033099 IP (tos 0x0, ttl 128, id 12687, len 40) 192.168.1.134.1026 > 192.168.1.128.53: R [tcp sum ok] **249746077:249746077**(0) ack **249751869** win 40982

So, what can we learn from this simple lab that applies to the analysis of the snort alert? Basically, that the attack is a real threat and it's pretty simple to accomplish. It's no fiction anymore. The snort detected traces have characteristics very similar to what was described to be necessary for this attack: lots of SYN packets, large window, destination port and address fixed. Even the fact of having a varying source port is OK, because we can imagine a scenario where the attacker would have to guess the source port as well. That would take more time, but is feasible. A single characteristic of the snort trace however, shows that it's in fact a common port scanning and a not reset attack: connection retries. The attacker sends three packets with same source port and sequential number. Repeating a sequence number using the same source/destination port number is not expected for this kind of attack as we learned from the lab. The same behavior is observed for source port 2844. See:

15:08:19.518648 10.10.10.195.**2834** > 172.20.11.80.179: S [tcp sum ok] **843877911**:**843877911**(0) win 64240 <mss 1460,nop,nop,sackOK> (DF) (ttl 128, id **38689**, len 48)

15:08:19.934638 10.10.10.195.**2834** > 172.20.11.80.179: S [tcp sum ok] **843877911**:**843877911**(0) win 64240 <mss 1460,nop,nop,sackOK> (DF) (ttl 128, id **38781**, len 48)

15:08:20.436254 10.10.10.195.**2834** > 172.20.11.80.179: S [tcp sum ok] **843877911**:**843877911**(0) win 64240 <mss 1460,nop,nop,sackOK> (DF) (ttl 128, id **38917**, len 48)

There are off course, other facts to consider:

a) Traces were captured on November 2003, few months before the paper and tools become available. It's not impossible, but unlikely that attacker had the knowledge and proper tools in hand to launch a RST attack at that time;

b) There are no legitimate connection between the victim and any other host using TCP 179 that would be a target for a spoofed RST attack at that time. The packets weren't even spoofed;

c) The target is not a router and it's wasn't running BGP;

d) And finally, there are lots of other signals in the captured file that shows this trace is part of vulnerability scanning launched against the target, probably using Nessus (but I am not going to prove that, my intention is to only prove this is a false positive).

Correlations:

My searches on Google and discussion lists haven't returned one single person relating similar detect. This fact is not a surprise, as the signature is pretty new as well the attack tools and technique.

All related RFCs can be found at IETF web site: http://www.ietf.org

The original Paul A. Watson's research paper can be found here:
http://www.packetstormsecurity.nl/papers/protocols/SlippingInTheWindow_v1.0.doc

NISCC Advisory is here:
http://*www.uniras.gov.uk/vuls/2004/236929/index.htm*

The reset tool is available at PacketStorm at following URL:
http://www.packetstormsecurity.nl/0404-exploits/reset.zip

CVE record for vulnerability:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0230

Evidence of active targeting:

The trace is part of a vulnerability assessment launched against the target and other machines in subnets 172.20.11.0/24, 172.11.11.0/24 and 172.10.11.0/24 by the attacker. The capture file shows that the attacker is actively targeting those machines. As a result of some sorting of port scan plugin, Snort has generated the alerts that were in fact, false positives.

Severity:

Severity is being evaluated using the following criteria:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Each item in the equation is given a number between 1 and 5, 1 being the lowest and 5 being the highest.

Criticality: if I guessed right, the target is a honeypot or a test lab computer with no critical function in the network, so I'll give 1 (just as an exercise, if this was in fact a border router I would give it 5)
Lethality: the attack in fact is a portscan phase of which appears to be a vulnerability assessment, there is no way to known if it was a legitimate scan done by security admin or by someone else, so I'll point the worst scenario that's 2 (in fact the scan had no observed collateral effect on the host).
System Countermeasures: no signal of any packet filtering in place at host level and no signal of any TCP/IP stack hardening to avoid OS fingerprinting as well, however no service was running at that port. I'll point 1 because no system countermeasure could be observed.
Network Countermeasures: no network level packet filtering device appears to be in place, however there is an IDS probe watching. I'll point 3.

Severity = (1 + 2) – (1 + 3) = -1

Defensive Recommendations:

Thais section is hard to evaluate, because I could not identify the real role (if any) of this computer in the network. All findings point to a honeypot or test lab computer, so how to secure that kind of equipment if they are there to be attacked? Supposing this in fact a network server, many security controls are recommended:

- Protect it with a packet filtering device at network perimeter
- Disable unneeded services
- Put access control list to allow only legitimate users to connect if possible (ex. TCPWrapers)
- Install latest patches and security updates from vendors
- Follow recommendations for hardening the Operating System as available at GIAC posted practicals web site (http://www.giac.org), SANS Reading Room (http://rr.sans.org) and SANS S.C.O.R.E (http://www.sans.org/score). That should

include hardening of TCP/IP stack and services banners to avoid fingerprinting and increase security against Denial of Services attacks like SYN Floods (or TCP Resets ☺).

Multiple choice question:

During a TCP session, what happens if one side receives a packet originated from the other side of connection containing an initial sequence number that falls within the range of a previously established session with that peer?

a) It accepts the ISN and proceeds with the three-way-handshake of a newer session
b) It accepts the ISN, but it thinks the packet belongs to that previously session and try to use it in that session
c) It silently drops the packet
d) It aborts the connection and informs user

Answer: d. RFC 793 states that.

I have sent this detect to the Intrusions mailing list on May 14, 2004, but unfortunately I haven't received any feedback. My post can be found at:
http://www.dshield.org/pipermail/intrusions/2004-May/007991.php

Network Detect 2 – Spoofed 127.0.0.1 Flood

Complete log entry:

04-22-2004 16:37:24 Local4.Critical X.Y.X.Y %PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.176.120 on interface inside

Log format:

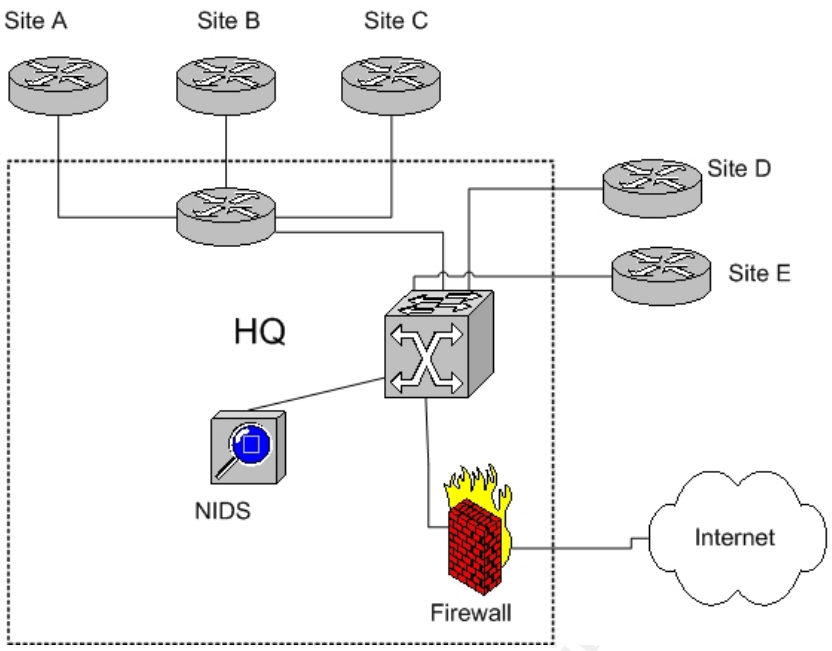
04-22-2004	Date
16:37:24	Time
Local4.Critical	Syslog facility.Syslog level
X.Y.X.Y	Firewall IP address
%PIX-2-106016	Log message

Firewall was generating thousands of spoof alert messages. Below is a sample of the generated alerts (simplified format to save space):

```
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.187.241 on interface inside
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.184.245 on interface inside
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.216.16 on interface inside
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.171.40 on interface inside
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.81.247 on interface inside
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.49.25 on interface inside
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.166.161 on interface inside
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.144.236 on interface inside
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.17.24 on interface inside
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.119.136 on interface inside
%PIX-2-106016: Deny IP spoof from (127.0.0.1) to 10.16.165.114 on interface inside
```

Source of Trace:

This trace was detected in the network of a company I did consulting for during a network incident. An approximated network diagram is shown below. Many other details, links and devices were removed for security reasons. I did some modifications in the topology for the same reason, but the main idea about the routing infrastructure was kept.



Routing is done in the multilayer switch at central site. Sites A, B and C have static default routes pointing to HQ's router, while Sites D and E connects directly to the switch. At each remote site, one or more L3 switches are used to spawn and route local VLANs. Routing between sites is done by HQ's multilayer switch. This one has a default route that points to firewall. A Network Intrusion Detection System was connected by me during the incident response process to a port of the switch that mirrors all traffic flowing between sites, as well the traffic that flow to and from the firewall. The stateful firewall has strong filtering rules that disallows any connection inbound. Outgoing rules allows only HTTP and HTTPS connections that comes from a proxy-cache device (not shown in the picture), that is also responsible for authenticating users.

Unfortunately, despite the good perimeter defense, most of remote sites failed to configure proper egress and ingress filtering, resulting in an inexistent defense against IP spoofing. So initially, there was no easy way to track down the attacker source. A tentative was made using techniques similar to what is described in Michael Gleen's GSEC practical paper "DoS/DDoS Prevention, Monitoring and Mitigation Techniques in a Service Provider Environment" available at:

http://www.giac.org/practical/GSEC/Michael_Glenn_GSEC.pdf.

This is an excellent and comprehensive paper on this subject and describes all commands and methodology utilized, so I'll consider that no complementary information about how I have tracked down the attacker's subnet is necessary. Netflow and ARP cache tables were investigated hop to hop until I got up to the source site (a VLAN of Site C). We get close to the attacker, but by whatever reason, the remote site's switch was not caching the ARP entry for 127.0.0.1. So I could no discover the MAC address of attacker's machine. A local tcpdump sniffer was placed at that site to help in the investigation and to collect forensics data.

Detect was generated by:

Detect was generated by a Cisco PIX firewall running PIX OS Version 6.3. The firewall logs all deny events to a central syslog server for storage, processing and visualization of logs.

Probability Source address was spoofed:

100% sure the source was spoofed. RFC 1122 states that:

"{ 127, <any> }

Internal host loopback address. Addresses of this form MUST NOT appear outside a host."

So, 127.0.0.1 is a reserved address to be only used for loopback connections. No packet should be sent out to the network using the loopback address, and in normal conditions no routing of packets containing source address equal to this reserved class should be performed, although this is not done by default. Filtering rules must be configured in the network gateways to performs that.

Description of Attack:

Quickly analysis of PIX logs points to a packet flood intended to create a Denial of Service condition on the network, by flooding it with spoofed packets. The directly effect of this attack was total unavailability of firewall, that had all its CPU resources consumed by these packets. However, many reports of similar activities were sent over some distribution lists around Internet. Analysis by Internet Storm Center showed that this was a result of Blaster worm. At time of Blaster worm attack many Internet Service Providers and network administrators have changed Microsoft Windows Update web site address to resolve to 127.0.0.1, so an infected host would attempt a denial of service against itself. As described by Internet Storm Center on handler's diary of March, 25th 2004, there is problem with this solution: if the infected host spoofs an IP address when connecting to 127.0.0.1 (SYN TCP DST 80) it attempts to respond to that spoofed address (RST SRC 80), generating the detected traces. I was aware of this ISC analysis and was able to recognize the pattern immediately. Checking if local administrators did the modifications on DNS or added any route policy I found no modification was made, but if this is Blaster, I should be able to find other traces, like signals of TCP 135 port scanning activities, which could help in tracking down the infected computer. In fact, after configuration of NIDS sensor at HQ's switch, I started seeing entries like the sample below:

[Snort Portscan2 preprocessor output]

```
04/22-16:37:24.439758  TCP src: 127.0.0.1 dst: 10.16.176.120 sport: 80 dport: 1121 flags: ***A*R**
04/22-16:37:24.440311  TCP src: 127.0.0.1 dst: 10.16.81.212 sport: 80 dport: 1289 flags: ***A*R**
04/22-16:37:24.460806  TCP src: 127.0.0.1 dst: 10.16.186.161 sport: 80 dport: 1119 flags: ***A*R**
04/22-16:37:24.471528  TCP src: 127.0.0.1 dst: 10.16.22.31 sport: 80 dport: 1854 flags: ***A*R**
04/22-16:37:24.481341  TCP src: 127.0.0.1 dst: 10.16.235.97 sport: 80 dport: 1937 flags: ***A*R**
04/22-16:37:24.501875  TCP src: 127.0.0.1 dst: 10.16.32.72 sport: 80 dport: 1620 flags: ***A*R**
04/22-16:37:24.502223  TCP src: 127.0.0.1 dst: 10.16.126.235 sport: 80 dport: 1452 flags: ***A*R**
04/22-16:37:24.522385  TCP src: 127.0.0.1 dst: 10.16.84.173 sport: 80 dport: 1535 flags: ***A*R**

<snip>
04/22-16:37:24.676152  TCP src: 10.16.24.132 dst: 156.190.248.81 sport: 4701 dport: 135 flags: ******S*
04/22-16:37:24.676267  TCP src: 10.16.24.132 dst: 156.190.248.82 sport: 4702 dport: 135 flags: ******S*
04/22-16:37:24.676389  TCP src: 10.16.24.132 dst: 156.190.248.83 sport: 4703 dport: 135 flags: ******S*
04/22-16:37:24.676503  TCP src: 10.16.24.132 dst: 156.190.248.84 sport: 4704 dport: 135 flags: ******S*
```

```
<snip>
04/22-16:37:24.677674  TCP src: 10.16.24.132 dst: 156.190.248.94 sport: 4714 dport: 135 flags: ******S*
04/22-16:37:24.677793  TCP src: 10.16.24.132 dst: 156.190.248.95 sport: 4715 dport: 135 flags: ******S*
04/22-16:37:24.677908  TCP src: 10.16.24.132 dst: 156.190.248.96 sport: 4716 dport: 135 flags: ******S*
```

Then, using the IP "10.16.24.132" to build a tcpdump filter, I could collect enough information to compare traffic coming from this source against traffic observed from "127.0.0.1" packets. After correlating the TTL and IPID fields of packets collected with tcpdump, I was almost sure they came from the same source computer. See the analysis of tcpdump data collected at remote location that confirms this assumption (filtering by source IP address and then by MAC):

```
WinDump.exe -r export-1.cap -vvne -c 1 ip src 10.16.24.132

16:37:25.142938 0:b:cd:13:82:18 0:b:5f:ec:f5:41 0800 62: IP (tos 0x0, ttl 128, id 53360,
len 48) 10.16.24.132.4705 > 156.190.248.85.135: S [tcp sum ok] 928173673:928173673(0) win
65535 <mss 1460,nop,nop,sackOK> (DF)

WinDump.exe -r export-1.cap -vvne ether src 0:b:cd:13:82:18
16:37:24.677915 0:b:cd:13:82:18 0:b:5f:ec:f5:41 0800 62: IP (tos 0x0, ttl 128, id 53314,
len 48) 10.16.24.132.4716 > 156.190.248.96.135: S [tcp sum ok] 928753506:928753506(0) win
65535 <mss 1460,nop,nop,sackOK> (DF)
16:37:24.689711 0:b:cd:13:82:18 0:b:5f:ec:f5:41 0800 60: IP (tos 0x0, ttl 128, id 53315,
len 40) 127.0.0.1.80 > 10.16.227.246.1716: R [tcp sum ok] 0:0(0) ack 1560936449 win 0
16:37:24.721019 0:b:cd:13:82:18 0:b:5f:ec:f5:41 0800 60: IP (tos 0x0, ttl 128, id 53318,
len 40) 127.0.0.1.80 > 10.16.73.156.1449: R [tcp sum ok] 0:0(0) ack 1041367041 win 0
```

Here we can observe a little jump from ID 53315 to 53318, this behavior repeats over time. The missing packets are probably the stimuli that generate the RST packets. As they were sent by the infected machine against itself, I was unable to detect it on the network.

Same process was repeated, until all infected computers were found. For our lucky, only five computers at that location were infected and all other location were free. The reason, later discovery, was an unofficial update from Windows 2000 Professional (that had all patches up to date) to Windows XP (factory defaults) by a help desk technician with the excuse of some laptop users were requesting it.

Attack Mechanism:

Analysis of worm's binary pointed to Blaster.E variant (file mslaugh.exe was found in the system's running process table). As any other previous variant, it exploits the DCOM RPC vulnerability using TCP 135 to spread. This variant will perform a DoS against Kimble.org rather than Windowsupdate.com. The problem is that kimble.org resolves to 127.0.0.1 (at time of this writing), see:

```
(C:>nslookup
Name server:  rio1.telemar.net.br
Address:  200.222.0.34

> kimble.org
Server:  rio1.telemar.net.br
Address:  200.222.0.34

Non-authorized answer:
Name =   kimble.org
Address:  127.0.0.1
```

That explains the spoofed RST packets even in absence of any local DNS modification. Analysis made by Symantec points out some of E's packets characteristics:

"The DoS traffic has the following characteristics:

- *Is a SYN flood on port 80 of kimble.org.*
- *Tries to send 50 HTTP packets every second.*
- *Each packet is 40 bytes in length.*
- *If the worm cannot find a DNS entry for kimble.org, it uses a destination address of 255.255.255.255.*

Some fixed characteristics of the TCP and IP headers are:

- *IP identification = 256*
- *Time to Live = 128*
- *Source IP address = a.b.x.y, where a.b are from the host ip and x.y are random. In some cases, a.b are random.*
- *Destination IP address = dns resolution of "kimble.org"*
- *TCP Source port is between 1000 and 1999*
- *TCP Destination port = 80*
- *TCP Sequence number always has the two low bytes set to 0; the two high bytes are random.*
- *TCP Window size = 16384"*

As we have only response packets, not all characteristics of detected packets will match with Symantec's description. For instance, we lost the IP identification and TCP Window size to compare, but some others do match. Look at following sample packets:

```
16:37:25.877417 IP (tos 0x0, ttl 128, id 53441, len 40) 127.0.0.1.80 > 10.16.27.160.1688:
R [tcp sum ok] 0:0(0) ack 1461452801 win 0
0x0000   4500 0028 d0c1 0000 8006 c55d 7f00 0001        E..(.......]....
0x0010   0a10 1ba0 0050 0698 0000 0000 571c 0001        .....P......W...
0x0020   5014 0000 ad1a 0000 0000 0000                  P.............
16:37:25.908564 IP (tos 0x0, ttl 128, id 53446, len 40) 127.0.0.1.80 > 10.16.128.71.1421:
R [tcp sum ok] 0:0(0) ack 941883393 win 0
0x0000   4500 0028 d0c6 0000 8006 60b1 7f00 0001        E..(......`.....
0x0010   0a10 8047 0050 058d 0000 0000 3824 0001        ...G.P......8$..
0x0020   5014 0000 6876 0000 0000 0000                  P...hv........
16:37:25.939737   IP   (tos   0x0,   ttl   128,   id   53448,   len   40)   127.0.0.1.80   >
10.16.229.109.1154: R [tcp sum ok] 0:0(0) ack 422313985 win 0
0x0000   4500 0028 d0c8 0000 8006 fb88 7f00 0001        E..(............
0x0010   0a10 e56d 0050 0482 0000 0000 192c 0001        ...m.P.......,..
0x0020   5014 0000 2353 0000 0000 0000                  P...#S........
```

- Length is 40 bytes
- Source port is always TCP 80 (in response to destination 80)
- Destination address is always at same Class B subnet as the infected host
- Destination port is always between 1000:2000 (in response to a source port falling in this range)
- Ack numbers have random high bytes and low bytes equals to 0001 (SEQ **RND**0000 increased by 1 as expected for a response of a stimulus packet containing SEQ number equals to **RND**0000)

Because most of destination addresses are directed at subnet not currently in use by the company, packets are being routed to the default route, which is in fact the firewall's internal IP address. This situation creates the Denial of Service against the firewall.

Correlations:

The ISC analysis of the 127.0.0.1 DoS is at:
http://isc.sans.org/diary.php?date=2004-03-25

Microsoft Advisory about RPC DCOM vulnerability:
http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx

Symantec's analysis of Blaster.E can be found at URL:
http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.e.worm.html

RFC 1122 is at:
http://www.ietf.org/rfc/rfc1122.txt?number=1122

Discussions about the 127.0.0.1 traffic:
http://cert.uni-stuttgart.de/archive/intrusions/2003/12/msg00102.html
http://cert.uni-stuttgart.de/archive/intrusions/2003/12/msg00106.html
http://www.dshield.org/pipermail/list/2004-January/029063.php
http://lists.freebsd.org/pipermail/freebsd-net/2004-March/003080.html

Analysis of Blaster worm and its traffic patterns done by other GIAC graduates:

http://www.giac.org/practical/GCIH/Sanjay_Menon_GCIH.pdf
http://www.giac.org/practical/GCIA/Greg_Bassett_GCIA.pdf
http://www.giac.org/practical/GCIA/Joanne_Schell_GCIA.pdf

You'll find information on Ingress and Egress filtering techniques at NSA Router Security Configuration Guide:
http://www.nsa.gov/snac/downloads_cisco.cfm?MenuID=scg10.3.1
Also at SANS Institute: http://www.sans.org/y2k/egress.htm

Michael Gleen's practical paper is available at:
http://www.giac.org/practical/GSEC/Michael_Glenn_GSEC.pdf

Evidence of active targeting:

Detected traces are a third party effect of a SYN flood directed to 127.0.0.1 with spoofed source IP address, thus resulting a flood of RST packets directed to the spoofed source addresses. No specific host or network was being targeted, although the original flood was directed against kimble.org.

Severity:

Severity is being evaluated using the following criteria:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Each item in the equation is given a number between 1 and 5, 1 being the lowest and 5 being the highest.

Criticality = I'll give 5. The company's firewall and all external connections were affected by this attack.

Lethality = I'll point 5 as well because the attack was successful in disrupting network communications at border level.

System Countermeasures = that's equal to 2, because no patches, personal firewalls or anti-virus were installed at infected computers. Although the worm failed to spread to other computers because all remaining hosts at company network were proper updated.

Network Countermeasures = I'll point 1, because no anti-spoofing filtering were in place at routers.

Severity = (5+5) – (2+1) = 7

Defensive Recommendations:

This is a good example of how a simple violation of the Security Policy can lead to a complete network disruption. Defensive recommendations must start at policy level. All company's employees should be aware of Security Policy and well trained to understand it well.

Extending of the security policy to include:

- No computer can be connected to corporate network prior to a complete update of its Operating System and services;
- All computers on the network must not connect to any portion of the company's network without proper and updated anti-virus software;
- All gateways devices must have anti-spoofing filtering enabled;

And if possible, all computers must have a host based Intrusion Detection system and a personal firewall. Network Intrusion Detection sensors installed around strategic network segments is also recommended.

Multiple choice question:

What's wrong with the following packet?

16:37:24.377366 IP (tos 0x0, ttl 128, id 53270, len 40) 127.0.0.1.80 > 10.16.228.172.1654: R [tcp sum ok] 0:0(0) ack 1109786625 win 0

a) RST packets should not have an acknowledgement number set
b) There is no TCP Sequential number
c) It violates RFC 1122
d) Window is zero

Answer: C. The address 127.0.0.1 is the loopback address and must not appear on the network. See RFC 1122 statement above.

Network Detect 3 – Gaobot knocks at my door.

```
1, 2004-04-23 20:31:43, 2003102, TCP_Probe_Other, 200.199.75.130, ,
200.199.37.187, , port=1025|2745|3127|5000|6129&reason=Firewalled, 5, A, 1881,
5000, 0x27406

1, 2004-04-23 20:56:25, 2003102, TCP_Probe_Other, 200.216.37.117,
RJ216037117.user.veloxzone.com.br, 200.216.123.171, ,
port=1025|2745|3127|6129&reason=Firewalled, 12, A, 4473, 6129, 0x27a06

1, 2004-04-24 14:21:31, 2003102, TCP_Probe_Other, 200.223.194.79, powernet-200-
223-194-79.atarde.com.br, 200.216.123.214, ,
port=1025|2745|3127|6129&reason=Firewalled, 6, A, 1300, 2745, 0x27206
```

Source of Trace:

These traces were collected at my home personal computer. This computer connects to internet trough a V90 modem dial-up connection. These are sample traces, collected from a log file containing hundreds of similar records.

Detect was generated by:

This detect was generated by Internet Security Systems BlackICE PC Protection version 3.6.cbx running on Windows XP box. The format of log is presented below. This description was taken from http://www.iss.net/security_center/advice/Support/KB/q000018/ knowledge base article:

> ***"What is the format of "attack-list.csv"?***
>
> *This article applies to: BlackICE Defender.*
>
> ***SUMMARY***
>
> *The file "attack-list.csv" contains the list of intrusions that the product found. The primary information lists the attack and the suspected intruder. This article explains the file format in more depth.*
>
> ***DETAILS***
>
> *This file is in "CSV" (Comma Separated Value) format, and can be imported into spreadsheets and database programs for further processing.*
>
> *The columns are, from left to right:*
>
> *severity*
> *This is a number from 1-99 that indicates the severity of an attack, where 1 is not very severe, and 99 is the most severe attack. Unfortunately, these levels do not have any precise meaning. Even an attack at level 1 may result in a compromise of the machine, whereas an attack at level 99 could be harmless. The assigned level is just a best-guess.*
> *timestamp*
> *This indicates the time and date of the **last** time the attack occurred. Attacks are "coalesced", meaning that if the same attack occurs multiple times, earlier attacks are sometimes removed from the list and simply merged with the latest one. A count of the number of times an attack has occurred is kept in another column. This timestamp is kept in GMT (aka UTC), and is probably several hours off from the time you see in the user interface. The ISP will want the time in this format so they don't have to worry about what timezone you are in.*
> *"issueId"*

A numeric identifier for this attack type. Each of the more than 300 attacks that the intrusion-detection component detects is assigned a unique number. This number is used for all internal processing of events. This number may also be pasted at the end of the URL http://advice.networkice.com/advice/intrusions/ in order to get help on the event.
"issueName"
The name of the attack. Each of the unique "issueId" numbers has a name associated with it.
intruder's IP address
The IP address of the attacker. Remember that IP addresses can sometimes be "spoofed" (forged), or that an intrusion may be a "false-positive", so there isn't a 100% chance that this is actually a hostile person.
intruder's name
The name of the intruder. We scan both Internet databases like DNS as well as the attacker itself in order to find the "best-name" of the machine, then display it here.
victim's IP address
This is the IP address of who the intruder was attacking. For example, if a user is running the product and gets attacked on a dial-up, then this will be the IP address assigned to that machine during that dialup session.
"parameters"
This contains some detailed information about the attack. For example, in a "TCP port probe" scan, this will contain a list of "ports" the attacker was scanning. The meaning of this information is documented in the "advICE" database.
count
The number of times this attack was seen."

Probability source address was spoofed:

Since this trace represents a worm scanning for possible targets, it's unlikely that source address is spoofed. In order to spread, the worm need to know which targets are available, so he needs to receive the responses of its stimuli.

Description of Attack:

Before describing the attack mechanism, I have to show how I concluded these traces were generated by a worm and not a scanning tool. First of all, they have purely random source address; they are not timely coordinated or even shows retries from the same source host. I did a search on Google for people reporting similar traces and found dozen of them saying they have caught the worm's binary from source host, which has showed to be an Agobot/Phatbot/Gaobot variant. The conclusive information I have used to form the basis of my analysis was taken from Internet Storm Center handler diary of 04-18-2004 available at http://isc.sans.org/diary.php?date=2004-04-18. So, comparing those descriptions with above traces we see they have enough matching characteristics to conclude it's a worm scanning activity.

This attack utilizes the worm's scanning capabilities to find vulnerable systems to compromise. The worm will then launch one of its exploits against the target in order to copy itself to the target machine. After infecting a vulnerable host, the worm starts scanning for other potential targets to infect. When scanning, the worm will search for hosts vulnerable to one of the following vulnerabilities (according to Symantec):

- Weak passwords on network shares.
- The DCOM RPC vulnerability (described in Microsoft Security Bulletin MS03-026) using TCP port 135.
- The WebDav vulnerability (described in Microsoft Security Bulletin MS03-007) using TCP port 80.

- The Workstation service buffer overrun vulnerability (described in Microsoft Security Bulletin MS03-049) using TCP port 445. Windows XP users are protected against this vulnerability if Microsoft Security Bulletin MS03-043 has been applied. Windows 2000 users must apply MS03-049.
- The Microsoft Messenger Service Buffer Overrun Vulnerability (described in Microsoft Security Bulletin MS03-043).
- The Locator service vulnerability (described in Microsoft Security Bulletin MS03-001) using TCP port 445. The worm specifically targets Windows 2000 machines using this exploit.
- The UPnP vulnerability (described in Microsoft Security Bulletin MS01-059).
- The vulnerabilities in the Microsoft SQL Server 2000 or MSDE 2000 audit (described in Microsoft Security Bulletin MS02-061), using UDP port 1434.
- The backdoor ports that the Beagle and Mydoom families of worms open.

This explains the ports being scanned. If successful in infecting a host, the process starts again. The worm also, opens a backdoor trough IRC so its creator can connect to the infected hosts and send commands to do any of the following commands (extracted from Symantec):

- Download and execute files
- Steal system information
- Send the worm to other IRC users
- Add new users accounts
- Perform Denial of Service attacks

Recent variants of this worm may add entries in the host's "hosts" file in order to disable access to certain antivirus and Windows Update websites. Symantec also states that the worm will try to terminate processes associated to antivirus and firewalls running on the infected host.

Attack Mechanism:

The log messages above shows some infected computers trying to detect if one of my machine's ports were opened, so he could launch one of its exploits. Fortunately, my computer was protected with a personal firewall that was able to block the scans, so the attacks didn't succeed.

Correlations:

Some posts of users detecting similar traces:
http://cert.uni-stuttgart.de/archive/intrusions/2004/03/msg00041.html
http://www.securityfocus.net/archive/75/363745/2004-05-17/2004-05-23/0
http://www.securityfocus.net/archive/75/363515/2004-05-17/2004-05-23/2

Blaine Hein sent a detect of which he concludes comes from one of Agobot/Phatbot exploits:
http://www.dshield.org/pipermail/intrusions/2004-April/007910.php

Microsoft Security Bulletin MS03-026:
http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx

Microsoft Security Bulletin MS03-007
http://www.microsoft.com/technet/security/bulletin/MS03-007.mspx

Microsoft Security Bulletin MS03-49
http://www.microsoft.com/technet/security/bulletin/MS03-049.mspx

Microsoft Security Bulletin MS03-043
http://www.microsoft.com/technet/security/bulletin/MS03-043.mspx

Microsoft Security Bulletin MS03-001
http://www.microsoft.com/technet/security/bulletin/MS03-001.mspx

Microsoft Security Bulletin MS01-059
http://www.microsoft.com/technet/security/bulletin/MS01-059.mspx

Microsoft Security Bulletin MS02-061
http://www.microsoft.com/technet/security/bulletin/MS02-061.mspx

Internet Storm Center analysis:
http://isc.sans.org/diary.php?date=2004-04-18

Symantec Analysis of Agobot family:
http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.gaobot.gen.html

I did a whois on some of source attacking hosts, but no relevant information has been found as most of then comes from dial-up users like me.

Evidence of active targeting:

These traces represent a scanning of randomly chosen destination addresses. No specific host or networks are being target.

Severity:

Severity is being evaluated using the following criteria:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Each item in the equation is given a number between 1 and 5, 1 being the lowest and 5 being the highest.

Criticality = I'll give 2. This is a desktop computer. It's important to me, but no critical resource or data would be lost in event of a catastrophe.

Lethality = I'll point 5 because the worm has a lot of potential exploits to compromise vulnerable hosts.

System Countermeasures = 5. Target host have an updated antivirus, all operating systems patches are applied and a Personal firewall with host IDS was in place.

Network Countermeasures = 5. It's difficult to consider a dial-up connection as a "network". So, considering there is a personal firewall/ids installed, I'll count 5.

Severity = (2+5) – (5+5) = -3

Defensive Recommendations:

The target computer has all necessary controls to defend against this threat, but for users that want to know what can be done to protect an un-protected computer, see the following recommendations:

- Keep the operating system updated with all available security patches and updates. Some OSes have a feature to periodically check for updates automatically. If your OS support this functionality, enable it.
- Install a good Antivirus software and keep it updated.
- Install a personal firewall and blocks unwanted ports. For instance, Windows XP already comes with a firewall functionality. Enable it in all of your untrusted interfaces.
- Consider purchasing a good Host based Intrusion Detection sensor.
- Install a good anti-spyware software, like Spybot - Search and Destroy.
- Don't open unknown mail attachments and avoid downloading files from suspicious websites.

Multiple choice question:

You have found that a certain host of your network is constantly and randomly scanning for the list of ports below. I have also, found a unknown binary and lots of associated process and ports, so you conclude it's probably a worm. From the list of scanned ports, what worm is more likely to be?

List of ports: TCP 135, 1025, 2745, 3127, 6129 and 5000.

a) One of Blaster variants
b) Welchia/Nachi
c) One of Phatbot/Agobot/Gaobot variant
d) Sasser worm.

Answer: C. Blaster scans for port TCP 135 and so do Welchia/Nachi (not considering the ICMP discovering scan it also does); Sasser scans for TCP port 445.

Part 3 – Analyze This

Intrusion Analysis Audit Report

Executive Summary

This report presents the results of an intense analysis of intrusion activities in the campus network of this University. The results of this analysis show that, comparing to previous reports, a significant improvement in the network security was achieved. It was detected a reduction in the number of complex attacks and exploits, although viruses stills a major problem. Many false positives due to Peer-to-Peer traffic are overwhelming the University's Intrusion Detection Systems, which makes the continual analysis of security alerts a hard process for your security staff I can suppose. So, it's extremely recommended that actions be taken to reduce the false positives caused by P2P. The likelihood of loosing something important within waves of false positives is currently significant.

Many internal hosts are compromised with viruses. Those computers represent a high risk for the University because they do not only affect the internal network, but they are also constantly scanning for external hosts, which may result in sues against the University.

At end of this report, some defensive recommendations are presented. Evaluation of recommended security controls is highly advised.

Analyzed Files

During this audit, five consecutive days of log files from an Intrusion Detection System were used. Those files were generated by a Snort sensor, placed in a strategic point of University's network, logging in "Fast" mode. In Fast mode, only a few set of information about the packets that triggered the alert are produced, thus limiting the data available for analysis. For instance a slightly modified version of snort Fast format was used, only the timestamp, alert description and address and port pairs are available in the Alert files, but we benefits from the performance gain of this log format. Two other categories of log files were also used: Out of Specification logs and Port Scan log files. The OOS files, in contrast with alert logs, do provide us with more detailed information about the packets captured, but they might not be related to any alert of Alert log files, so correlation is not always possible. OOS alerts are associated to violations of protocols, packet corruption or bad TCP/IP stack implementations. The Ports Scan files contain traces of portscanning activities detected in the network. The University doesn't provide us with any information regarding how the portscan preprocessor that generated the logs has been tuned, so we might be dealing with lots of false positives (i.e. normal network conditions). A characteristic of snort portscanning alerting mechanism is that, for a certain threshold of records it writes to the portscan file, it also produces an alert entry in the Alert files, so we can remove those portscanning alerts from the Alerts files without loosing anything.

Here comes the list of files utilized during this analysis. The files were downloaded from the Internet Storm Center repository at: http://www.incidents.org/logs/ .

```
$ ls -o
total 150567
-rwx------+   1 JC        1714968 May 17 10:26 alert.040407.gz
-rwx------+   1 JC        3918959 May 17 10:29 alert.040408.gz
```

```
-rwx------+   1 JC        4134815 May 17 10:32 alert.040409.gz
-rwx------+   1 JC        5008515 May 17 10:35 alert.040410.gz
-rwx------+   1 JC        4977930 May 17 10:38 alert.040411.gz
-rwx------+   1 JC        3456000 May 17 12:17 oos_report_040407.log
-rwx------+   1 JC        1341440 May 17 12:19 oos_report_040408.log
-rwx------+   1 JC         516096 May 17 12:20 oos_report_040409.log
-rwx------+   1 JC        1638400 May 17 12:22 oos_report_040410.log
-rwx------+   1 JC         360448 May 17 12:24 oos_report_040411.log
-rwx------+   1 JC       28569712 May 17 10:56 scans.040407.gz
-rwx------+   1 JC        8937472 May 17 11:08 scans.040408.gz
-rwx------+   1 JC       21184512 May 17 12:10 scans.040409.gz
-rwx------+   1 JC       41031562 May 17 12:52 scans.040410.gz
-rwx------+   1 JC       27385856 May 17 12:52 scans.040411.gz
```

Files of same type were merged to facilitate the analysis process (discussed in the appendix section). As pointed out by other students, replacing MY.NET from the log files with a real numeric network representation, facilitates the handling of data using databases and UNIX text utilities. So, I replaced MY.NET with 130.85 using UNIX *sed* utility (details in analysis process section).

Summary of Findings from Alert Files

This section provides a summary of alerts found the Alert log files. They are sorted by number of occurrence. In the next section, the top 10 critical alerts will be analyzed separately.

Alert	*# Occurrences*
EXPLOIT x86 NOOP	28826
130.85.30.3 activity	12996
SMB Name Wildcard	12173
High port 65535 tcp - possible Red Worm - traffic	10664
130.85.30.4 activity	10207
Tiny Fragments - Possible Hostile Activity	8014
DDOS mstream handler to client	3262
Null scan!	1126
NMAP TCP ping!	1098
Possible trojan server activity	1081
External RPC call	930
SUNRPC highport access!	637
Incomplete Packet Fragments Discarded	511
TCP SRC and DST outside network	309
High port 65535 udp - possible Red Worm - traffic	244
ICMP SRC and DST outside network	210
[UMBC NIDS] Internal MiMail alert	158
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	147
DDOS shaft client to handler	142
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	108
FTP passwd attempt	100
TCP SMTP Source Port traffic	83
IRC evil - running XDCC	72
EXPLOIT x86 setuid 0	66
SMB C access	55
[UMBC NIDS] External MiMail alert	47

connect to 515 from outside	46
EXPLOIT x86 setgid 0	33
EXPLOIT x86 stealth noop	28
[UMBC NIDS IRC Alert] Possible drone command detected.	25
RFB - Possible WinVNC - 010708-1	24
FTP DoS ftpd globbing	22
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	17
NIMDA - Attempt to execute cmd from campus host	15
TFTP - Internal UDP connection to external tftp server	14
Attempted Sun RPC high port access	14
SYN-FIN scan!	13
EXPLOIT NTPDX buffer overflow	10
EXPLOIT x86 NOPS	8
DDOS mstream client to handler	6
Probable NMAP fingerprint attempt	6
TFTP - External TCP connection to internal tftp server	4
NETBIOS NT NULL session	3
PHF attempt	2
[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	2
[UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan.	2
External FTP to HelpDesk 130.85.70.50	1
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	1
External FTP to HelpDesk 130.85.70.49	1
Fragmentation Overflow Attack	1
External FTP to HelpDesk 130.85.53.29	1

Analysis of TOP 10 Alerts

Prioritized by Severity, and then by number of occurrences.

Alert	*Severity*	*# Occurrences*
EXPLOIT x86 NOOP	1	28826
IRC evil - running XDCC	1	72
EXPLOIT x86 setuid 0	1	66
EXPLOIT x86 setgid 0	1	33
EXPLOIT x86 stealth noop	1	28
NIMDA - Attempt to execute cmd from campus host	1	15
EXPLOIT NTPDX buffer overflow	1	10
Possible trojan server activity	2	1081
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	2	147
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	2	108
FTP DoS ftpd globbing	2	22

TOP 1 – "EXPLOIT x86 NOOP"

NO OPeration is a very common technique used in development of software exploits.

NOPs are used to pad data into a buffer with intention of overflow it. The right use of NOPs and other low level instructions will allow an attacker to inject malicious code into application buffer. If done right, the buffer overflow will result in the execution of inserted malicious code. Unfortunately, the signature that triggers this alert is susceptible to a high rate of false positives, since NOPs can be detected in the contents of any binary file, like a JPEG image for example, that a host might be downloading. So, without packet dumps, there is no way to determine if this false positive or not, however my feeling points to a false positive, since a great number of hosts, involved in very distinct kinds of conversations, are associated to this signature. As an example, the table below shows alerts associated to HTTP traffic:

```
04/07-14:13:12.169876 [**] EXPLOIT x86 NOOP [**] 199.131.21.34:2724 -> 130.85.112.209:80
04/07-14:13:13.103787 [**] EXPLOIT x86 NOOP [**] 199.131.21.34:3149 -> 130.85.84.135:80
04/07-14:13:13.103920 [**] EXPLOIT x86 NOOP [**] 199.131.21.34:3149 -> 130.85.84.135:80
04/07-14:13:13.104758 [**] EXPLOIT x86 NOOP [**] 199.131.21.34:3149 -> 130.85.84.135:80
04/07-14:13:13.105053 [**] EXPLOIT x86 NOOP [**] 199.131.21.34:3149 -> 130.85.84.135:80
04/07-14:13:13.107248 [**] EXPLOIT x86 NOOP [**] 199.131.21.34:3149 -> 130.85.84.135:80
04/10-18:16:47.548340 [**] EXPLOIT x86 NOOP [**] 68.43.170.140:3879 -> 130.85.17.4:80
04/10-18:16:47.709059 [**] EXPLOIT x86 NOOP [**] 68.43.170.140:3879 -> 130.85.17.4:80
04/10-18:16:47.803320 [**] EXPLOIT x86 NOOP [**] 68.43.170.140:3879 -> 130.85.17.4:80
04/10-18:16:47.851702 [**] EXPLOIT x86 NOOP [**] 68.43.170.140:3879 -> 130.85.17.4:80
```

HTTP traffic is a common source of false positives for buffer overflows signatures, because lots of figures and files are downloaded constantly. This is so problematic, that default Snort Configuration file, exclude this port from the ports where it will look for buffer overflows. See:

```
# Ports you want to look for SHELLCODE on.
var SHELLCODE_PORTS !80
```

The above line tells snort to not use traffic on port 80 when evaluating shellcode rules. My conclusion, in absence of full packet captures, is that these alerts can be safely ignored when associated to HTTP traffic. Pete Storm's analysis of similar traffic (see the following URL http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf), points to same interpretation, i.e., that these alerts are false alerts produced by a transferring of a binary stream.

Top Sources:

Source	*Count*	*Total Alerts*	*# Dsts (sig)*	*# Dsts (total)*
199.131.21.34	3480	3481	499	500
68.43.170.140	1488	1566	85	87
67.113.214.132	691	691	53	53
200.205.95.10	467	467	58	58
61.32.236.202	466	466	145	145

TOP 2 – "IRC evil - running XDCC"

XDCC is file share mechanism utilized by IRC (internet relay chats) users to distribute software to each other. This alert must be considered very suspicious, and the machines 130.85.43.2, 130.85.82.79 and 130.85.43.7 fully investigated for signals of any compromise. This is because 1) XDCC felt in disuse after popularization of P2P networks, like GNUTELLA, as a way of "innocent" file sharing; 2) The large widespread of recent viruses equipped with IRC bots, like "phatbot", has been used to create large networks of

"zombies" machines, ready to be used for *warez*, spamming and other nefarious purposes. The likelihood of false positives for these alerts are very low, as the hosts involved are also associated to other alerts related to IRC *bot'ing*. See table below.

Top Sources

Source	*Count*	*Correlations (Other alerts associated to this source)*
130.85.43.2	60	*Null scan!* *EXPLOIT x86 setgid 0* *[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot* *[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.* *High port 65535 udp - possible Red Worm - traffic* *[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.* *Tiny Fragments - Possible Hostile Activity*
130.85.82.79	11	*EXPLOIT x86 setuid 0* *SYN-FIN scan!* *[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.* *[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.* *High port 65535 tcp - possible Red Worm - traffic* *Null scan!*
130.85.43.7	1	None

The correlations would explain most of occurrences of other alerts. They are certainly the result of infected internal machines scanning for active hosts in the network in order to launch other attacks.

Top Destinations

Destinations	*Count*	*Correlations (Other alerts associated to this destination)*
64.246.60.72	60	*[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.*
207.36.180.241	11	*[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.*
64.62.196.26	1	*[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.*

TOP 3/5 – "EXPLOIT x86 setuid 0", "EXPLOIT x86 setgid 0" and "EXPLOIT x86 stealth noop"

Analysis of these alerts point to same interpretation of "Exploit x86 NOOP" alerts and P2P traffic for most occurrences, however few of then requires immediate action, as they are almost sure related to worm activity, see:

```
04/08-07:09:25.855507 [**] EXPLOIT x86 setgid 0 [**] 131.175.65.35:1189 ->
                                                          130.85.75.88:2002
04/08-07:26:35.844276 [**] EXPLOIT x86 setuid 0 [**] 131.175.65.35:1203 ->
                                                          130.85.75.88:2002
04/08-07:32:51.649735 [**] EXPLOIT x86 setuid 0 [**] 131.175.65.35:1210 ->
                                                          130.85.75.88:2002
04/08-09:40:57.209007 [**] EXPLOIT x86 setuid 0 [**] 131.175.65.35:1526 ->
                                                          130.85.75.88:2002
```

```
04/09-04:25:23.500293 [**] EXPLOIT x86 setuid 0 [**] 142.166.83.35:42416 ->
                                                              130.85.75.88:2002
04/09-05:38:25.973099 [**] EXPLOIT x86 setuid 0 [**] 142.166.83.35:44238 ->
                                                              130.85.75.88:2002
```

Port 2002 TCP is associated to TransCout Trojan and 2002 UDP to Slapper P2P worm backdoor. Default Snort rule for this alert will trigger in presence of both protocols.

```
"alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE x86 setgid 0";
content: "|b0b5 cd80|"; reference:arachnids,284; classtype:system-call-detect; sid:649;
rev:6;)"
```

Without more information there is no way to determine if these alerts are related to one or both worms.

TOP 6 – "NIMDA - Attempt to execute cmd from campus host"

It looks like this detect has been triggered by a customized rule. It alerts when internal machines tries to execute "cmd.exe" on external IIS web servers. This behavior is related to exploitation of Microsoft IIS Unicode vulnerability, utilized by NIMDA worm to infect other hosts. Sources of this signature are probably infected and actively scanning for targets, however there are only few occurrences of this signature. That makes me consider the possibility of use of an automated vulnerability scanning tool instead of worm propagation. I recommend that source machines be disconnected from the network and fully analyzed. If infection is confirmed, rebuild the machine from scratch and apply all patches.

Source	*Count*
130.85.97.228	3
130.85.97.25	2
130.85.97.166	2
130.85.17.45	2
130.85.97.74	2
130.85.97.180	1
130.85.97.69	1
130.85.97.36	1
130.85.10.79	1

Source Hosts

Searches in portscan files for scans against port 80 having these machines as source hosts haven't returned anything for most hosts. This might be an indicative of false positive, since we could expect to see traces of NIMDA scans on those files, as showed by Ian Martin in his GCIA paper http://www.sans.org/rr/papers/index.php?id=1128. I was able to confirm the scanning activity only for 130.85.97.74 host:

```
Apr 11 00:53:02 130.85.97.74:1401 -> 64.185.226.99:80 SYN ******S*
Apr 11 00:53:02 130.85.97.74:1402 -> 195.248.190.32:80 SYN ******S*
Apr 11 00:53:02 130.85.97.74:1403 -> 195.161.119.248:80 SYN ******S*
Apr 11 00:53:02 130.85.97.74:1404 -> 217.23.142.153:80 SYN ******S*
Apr 11 00:53:04 130.85.97.74:1425 -> 217.23.142.153:8080 SYN ******S*
Apr 11 00:53:02 130.85.97.74:1412 -> 217.16.19.225:80 SYN ******S*
Apr 11 00:53:02 130.85.97.74:1413 -> 195.161.116.65:80 SYN ******S*
Apr 11 00:53:02 130.85.97.74:1414 -> 62.149.0.132:80 SYN ******S*
Apr 11 00:53:02 130.85.97.74:1415 -> 81.19.66.19:80 SYN ******S*
Apr 11 00:53:02 130.85.97.74:1416 -> 217.16.16.110:80 SYN ******S*
Apr 11 00:53:02 130.85.97.74:1417 -> 62.118.240.78:80 SYN ******S*
Apr 11 00:53:03 130.85.97.74:1418 -> 64.185.226.101:80 SYN ******S*
```

TOP 7 – EXPLOIT NTPDX buffer overflow

This alert was triggered by the following snort signature:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 123 (msg:"EXPLOIT ntpdx overflow
attempt"; dsize: >128; reference:arachnids,492; reference:bugtraq,2540;
classtype:attempted-admin; sid:312; rev:2;)
```

This attack is a buffer overflow exploit against a buggy version of Network Time Protocol

daemon that affected many platforms. Bugtraq reference for this vulnerability can be found at: http://www.securityfocus.com/bid/2540

Note that this vulnerability is not an indicative of successful compromise. It only alerts that an exploit was attempted. We need further investigate if one really has succeeded. I'll try to correlate the attacked hosts as sources of other alerts. That might indicate a compromised host.

Destinations	*# Count for this signature*	*# Total alerts for this host*
130.85.66.29	2	5
130.85.84.234	2	55
130.85.6.62	2	2
130.85.16.106	1	9
130.85.84.133	1	18
130.85.97.60	1	6
130.85.97.83	1	3

From the above list, the only host that is also source for other alerts is host 130.85.6.62. Here comes the stimuli attacks that tries to exploit the ntpd vulnerability:

```
04/10-18:04:19.144948 [**] EXPLOIT NTPDX buffer overflow [**] 69.140.137.209:100 ->
130.85.6.62:123
04/10-18:04:19.555998 [**] EXPLOIT NTPDX buffer overflow [**] 69.140.137.209:100 ->
130.85.6.62:123
```

And here the same host acts as a source for Red Worm traffic:

```
04/07-21:23:42.837448 [**] High port 65535 udp - possible Red Worm - traffic
[**] 130.85.6.62:65535 -> 69.140.137.209:65280
04/07-21:23:42.900873 [**] High port 65535 udp - possible Red Worm - traffic
[**] 130.85.6.62:65535 -> 69.140.137.209:65280
04/07-21:23:48.688887 [**] High port 65535 udp - possible Red Worm - traffic
[**] 130.85.6.62:65535 -> 69.140.137.209:65280
```

Careful analysis of timeline shows that these alerts don't correlate. The exploit against ntpd was launched after signals of other malicious activity were detected. It's a coincidence both types of alert be related to same host but unlikely they are related to same attack.

In his analysis, Anton Chuvakin (http://www.giac.org/practical/GCIA/Anton_Chuvakin_GCIA.pdf) has found that similar alerts of NTPDX overflow were in fact, mostly false positives caused by P2P traffic. I do agree with his conclusions.

TOP 8 – Possible trojan server activity

This alerts when traffic directed to port 27374 is detected on network. A query in neohapsis database shows that many Trojan horses make use of this port to communicate:

Protocol	Service	Name
tcp	SubSeven	[trojan] SubSeven
tcp	SubSeven	[trojan] SubSeven
tcp	BadBlood	[trojan] Bad Blood
tcp	EGO	[trojan] EGO
tcp	FakeSubSeven	[trojan] Fake SubSeven
tcp	Lion	[trojan] Lion
tcp	Ramen	[trojan] Ramen
tcp	Seeker	[trojan] Seeker
tcp	Subseven2.1.4DefCon8	[trojan] Subseven 2.1.4 DefCon 8
tcp	SubSeven2.1Gold	[trojan] SubSeven 2.1 Gold
tcp	SubSeven2.2	[trojan] SubSeven 2.2
tcp	SubSevenMuie	[trojan] SubSeven Muie
tcp	TheSaint	[trojan] The Saint
tcp	Ttfloader	[trojan] Ttfloader
tcp	Webhead	[trojan] Webhead

This alert is being triggered by 41 different sources and directed to 315 destinations. This is a very high rate of occurrences. We must consider that signatures based on port numbers are very susceptible to false positives, especially in these days of P2P networking.

One of the hosts involved with these alerts, is also involved with some other malicious traffic. The host 130.85.84.235 must be disconnected from the network immediately and full security analysis performed. This host is also involved with following alerts:

As source:

- *[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC*
- *High port 65535 tcp - possible Red Worm - traffic*
- *DDOS mstream handler to client*

As destination:

- EXPLOIT x86 setuid 0
- DDOS mstream client to handler
- Incomplete Packet Fragments Discarded
- Null scan!
- DDOS shaft client to handler
- High port 65535 tcp - possible Red Worm - traffic

This is a very suspicious host. Portscanning activity shows this host is high involved with P2P traffic as well:

```
Apr  9 05:54:03 130.85.84.235:24320 -> 83.33.210.57:4662 SYN ******S*
Apr  9 05:54:03 130.85.84.235:24321 -> 81.36.110.228:4662 SYN ******S*
Apr  9 05:54:03 130.85.84.235:24322 -> 82.223.21.105:4662 SYN ******S*
Apr  9 05:54:03 130.85.84.235:24323 -> 80.38.233.26:4664 SYN ******S*
Apr  9 05:54:03 130.85.84.235:24324 -> 80.36.206.44:4662 SYN ******S*
Apr  9 05:54:03 130.85.84.235:24333 -> 81.34.18.199:4662 SYN ******S*
Apr  9 05:54:03 130.85.84.235:24326 -> 81.37.55.93:4662 SYN ******S*
Apr  9 05:54:04 130.85.84.235:24335 -> 217.125.147.7:4662 SYN ******S*
```

```
Apr  9 05:54:03 130.85.84.235:24328 -> 83.32.22.81:4662 SYN ******S*
Apr  9 05:54:04 130.85.84.235:24313 -> 81.35.111.17:4662 SYN ******S*
Apr  9 05:54:05 130.85.84.235:24324 -> 80.36.206.44:4662 SYN ******S*
Apr  9 05:54:06 130.85.84.235:24192 -> 217.106.18.150:4665 UDP
Apr  9 05:54:06 130.85.84.235:24323 -> 80.38.233.26:4664 SYN ******S*
Apr  9 05:54:06 130.85.84.235:24328 -> 83.32.22.81:4662 SYN ******S*
Apr  9 05:54:06 130.85.84.235:24321 -> 81.36.110.228:4662 SYN ******S*
```

Ports 4662, 4664 and 4665 are related to eMule, eDonkey and other P2P applications. Many occurrences of the following traffic are also observed:

```
Apr 11 12:54:23 130.85.84.235:26704 -> 2.1.0.85:13 UDP
Apr 11 12:54:28 130.85.84.235:26716 -> 2.1.0.85:13 UDP
Apr 11 13:39:51 130.85.84.235:18788 -> 2.1.0.85:12 UDP
Apr 11 13:39:56 130.85.84.235:18813 -> 2.1.0.85:12 UDP
Apr 11 13:40:09 130.85.84.235:18864 -> 2.1.0.85:15 UDP
Apr 11 13:43:48 130.85.84.235:19772 -> 2.1.0.85:13 UDP
Apr 11 13:43:50 130.85.84.235:19793 -> 2.1.0.85:13 UDP
Apr 11 13:43:52 130.85.84.235:19793 -> 2.1.0.85:13 UDP
Apr 11 13:43:53 130.85.84.235:19793 -> 2.1.0.85:13 UDP
Apr 11 13:44:02 130.85.84.235:19834 -> 2.1.0.85:13 UDP
Apr 11 14:30:29 130.85.84.235:12268 -> 2.1.0.85:15 UDP
Apr 11 14:30:32 130.85.84.235:12289 -> 2.1.0.85:15 UDP
```

This is an anomalous traffic and should be investigated. As a side note, UDP port 13 is associated to daytime protocol, 12 and 15 are unregistered. Searches on Dshield.org for vulnerabilities on these ports haven't returned anything, so it's probably noise. Analysis from Tom King and Doug Kites cover similar detects.

TOP 9 – [UMBC NIDS IRC Alert] IRC user /kill detected, possible Trojan
TOP 10 – [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC

These alerts are related to IRC bot activity. Viruses containing backdoors based on IRC protocol has become very common, since popularization and release of source code of *phatbot* and others. After a host is compromised by phatbot/agobot or similar variants, a backdoor that allows attackers to take control of the system remotely will be available. The virus spreads like a worm, searching for targets to infect using various exploits for Windows vulnerabilities. For example, see Symantec's description of how Gaobot variant spreads over the Internet:

"W32.hllw.Gaobot is a worm that spreads through open network shares, backdoors that the Beagle and Mydoom worms install, and several Windows vulnerabilities, including:

- *Weak passwords on network shares.*
- *The DCOM RPC vulnerability:http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx*
- *The WebDav vulnerability:http://www.microsoft.com/technet/security/bulletin/MS03-007.mspx*
- *The Workstation service buffer overrun vulnerability: http://www.microsoft.com/technet/security/bulletin/MS03-049.mspx*
- *The Microsoft Messenger Service Buffer Overrun vulnerability: http://www.microsoft.com/technet/security/bulletin/MS03-043.mspx*
- *The Locator service vulnerability: http://www.microsoft.com/technet/security/bulletin/MS03-001.mspx*
- *The UPnP vulnerability: http://www.microsoft.com/technet/security/bulletin/MS01-059.mspx*
- *The vulnerabilities in the Microsoft SQL Server 2000 or MSDE 2000 audit:http://www.microsoft.com/technet/security/bulletin/MS02-061.mspx*
- *The backdoor ports that the Beagle and Mydoom families of worms open.*

It also opens backdoors to the infected computers through IRC.
http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.gaobot.gen.html

The worm can also act as a backdoor server program and attack other systems. Additionally, the worm attempts to stop the process of many antivirus and security programs".

The IRC bot portion of the virus will connect to pre configured IRC Server and channel, so it can be found by its creator. The attacker will use an IRC client to connect to the backdoor and send commands to the infected host, using it as a kind of zombie. Those commands can instruct the infected host to flood the local network to create a Denial of service, to portscan a specific host and others.

It's imperative that all internal hosts involved with these alerts be disconnected from the network and rebuilt.

There are a total of 47 destinations for "IRC user /kill" alert and 17 sources for "sdbot floodnet". From this amount, the list of those that are also involved with other alerts and should be considered a security priority:

IRC Trojan alert correlation table

Top Attacker	Top Destination	Destination is source for:
128.122.66.204	130.85.112.152	• *[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC* • *SMB Name Wildcard*
	130.85.150.199	• *[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC*
	130.85.80.224	• *[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC* • *[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC*
	130.85.5.44	*None*
	130.85.80.5	• *[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC* • *[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC*
	130.85.60.40	*None*
	130.85.43.2	• *IRC* evil - *running XDCC*

A signature that proves a host is infected with one of *bot virus is its scanning pattern. According to Internet Storm Center (http://isc.sans.org/diary.php?date=2004-04-18), a host infected with phatbot/gaobot/agobot will scan the network for ports 2745, 3127, 6129 and others. Putting this information on the analysis process, I did a search on portscan log for signals of similar activity, so I could prove the infection. Below are sample scanning activity for two of reported hosts:

```
Apr  7 16:03:31 130.85.112.152:2271 -> 130.117.69.227:3127 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2272 -> 130.117.69.227:6129 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2273 -> 130.117.69.227:139 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2274 -> 130.117.69.227:80 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2305 -> 130.120.51.167:3127 SYN ******S*
```

```
Apr  7 16:03:31 130.85.112.152:2306 -> 130.120.51.167:6129 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2307 -> 130.120.51.167:139 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2308 -> 130.120.51.167:80 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2309 -> 130.8.12.228:2745 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2310 -> 130.8.12.228:135 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2311 -> 130.8.12.228:1025 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2312 -> 130.8.12.228:445 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2313 -> 130.8.12.228:3127 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2315 -> 130.8.12.228:6129 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2316 -> 130.8.12.228:139 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2317 -> 130.8.12.228:80 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2318 -> 130.135.245.185:2745 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2319 -> 130.135.245.185:135 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2320 -> 130.135.245.185:1025 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2321 -> 130.135.245.185:445 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2322 -> 130.135.245.185:3127 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2323 -> 130.135.245.185:6129 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2324 -> 130.135.245.185:139 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2325 -> 130.135.245.185:80 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2326 -> 130.94.173.89:2745 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2327 -> 130.94.173.89:135 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2328 -> 130.94.173.89:1025 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2329 -> 130.94.173.89:445 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2330 -> 130.94.173.89:3127 SYN ******S*
Apr  7 16:03:31 130.85.112.152:2331 -> 130.94.173.89:6129 SYN ******S*

<snip>

Apr  7 15:20:47 130.85.150.199:3915 -> 130.1.36.160:445 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3916 -> 130.1.36.160:3127 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3917 -> 130.1.36.160:6129 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3918 -> 130.1.36.160:139 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3919 -> 130.1.36.160:3410 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3920 -> 130.1.36.160:5000 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3921 -> 130.59.227.16:2745 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3922 -> 130.59.227.16:135 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3923 -> 130.59.227.16:1025 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3924 -> 130.59.227.16:445 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3925 -> 130.59.227.16:3127 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3926 -> 130.59.227.16:6129 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3927 -> 130.59.227.16:139 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3928 -> 130.59.227.16:3410 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3929 -> 130.59.227.16:5000 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3947 -> 130.251.88.69:2745 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3948 -> 130.251.88.69:135 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3949 -> 130.251.88.69:1025 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3950 -> 130.251.88.69:445 SYN ******S*
Apr  7 15:20:47 130.85.150.199:2213 -> 130.133.25.3:2745 SYN ******S*
Apr  7 15:20:47 130.85.150.199:3972 -> 130.66.42.251:2745 SYN ******S*
Apr  7 15:20:47 130.85.150.199:4126 -> 130.251.246.240:2745 SYN ******S*
```

As an update for the referenced Symantec note, the worm also scans for ports 6129 and 5000. Port 6129 is used for Dameware and 5000 is believed to be used by the host to identify Windows XP machines, so proper shellcode can be used. Dameware suffers form a buffer overflow vulnerability that allows remote execution of malicious code. More information at: http://www.securityfocus.com/bid/9213.

Attention: Hosts listed as destinations in the above table are all infected and actively scanning the network for targets. Disconnect them from the network immediately and rebuild them from scratch.

Other Events of Interest

In the previous section I presented an analysis of ten most critical alerts I've categorized. Because of time and size constraints I am not going to discuss all of detected alerts, however there are two other alerts not in the list that deserves an especial attention:

- TFTP - Internal UDP connection to external tftp server
- RFB - Possible WinVNC - 010708-1

TFTP is a very common way worms utilize to download malicious content and VNC is remote administration software we should consider suspicious if not used by network administrators.

Alert	*Host*	*Count*
TFTP - Internal UDP connection to external tftp server	130.85.70.225	5
	130.85.111.34	1
	130.85.60.16	1
RFB - Possible WinVNC - 010708-1	130.85.111.51	3
	130.85.111.46	2
	130.85.84.231	2
	130.85.70.225	2
	130.85.53.44	1
	130.85.53.31	1

Port Scanning Activities

My analysis of scanning activity will start with some useful statistics. This is required in order to plot a big picture of hosts involved. Table below shows TOP 10 source hosts scanning the network:

TOP 10 Scanners	
Count	**Host**
2890299	130.85.1.3
1623363	130.85.111.51
1522547	130.85.153.35
1189493	130.85.81.39
1130435	130.85.70.96
1082054	130.85.112.152
796650	130.85.1.4
338203	130.85.66.56
295221	130.85.84.235
253164	130.85.42.2

Next table show top 10 list of targets following by top 10 destination ports:

TOP 10 Target	
Count	**Host**
107462	69.6.57.4
89844	69.6.57.7
89665	69.6.57.9
68502	192.26.92.30
55518	192.48.79.30
51601	130.85.60.38
46567	192.5.6.30
45740	69.6.57.8
45541	69.6.57.10
44676	195.228.156.17

TOP 10 Destination Ports	
Count	**Port**
3666094	53 (UDP)
3301898	135
754845	25
565438	2745
555423	80
550329	6129
467080	3127
462340	445
449900	1025
415404	139

Interesting to observe top scanned port is port 53, being 130.85.1.3 and 130.85.1.4 the top scanners for this port. A question arrives from this observation: is this just a result of extremely loaded DNS servers doing recursions for internal clients or a result of two compromised computers scanning for possible targets? If we accepted that internal network is fully loaded with P2P users, and suppose that those P2P client applications would try to resolve hostnames of its connected peers, we should expect to see so randomly DNS traffic. The query below confirms that both machines are DNS servers:

```
# nslookup
> set type=mx
> umbc.edu
Server: localhost
Address: 127.0.0.1
umbc.edu preference = 10, mail exchanger = mxin.umbc.edu
umbc.edu nameserver = UMBC5.umbc.edu
umbc.edu nameserver = UMBC3.umbc.edu
umbc.edu nameserver = UMBC4.umbc.edu
mxin.umbc.edu internet address = 130.85.12.6
UMBC5.umbc.edu internet address = 130.85.1.5
UMBC3.umbc.edu internet address = 130.85.1.3
UMBC4.umbc.edu internet address = 130.85.1.4
```

If this not the case, consider these server infected with a worm and actively scanning for DNS servers to propagate. Lion and ADMworm infect vulnerable DNS servers and propagate trough port 53. For information about these worms see http://www.sophos.com/virusinfo/analyses/unixadmworm.html and http://www.f-secure.com/v-descs/lion.shtml.

The majority of past posted practicals from other GIAC students also points that these addresses were suspiciously scanning for port 53 UDP. This makes me consider that this is normal but huge DNS activity, otherwise the University administrators are not aware of those analyses.

Scans for destination port 135 are caused by machines compromised by worms that exploit the eternal Microsoft RPC DCOM vulnerability, like Blaster. A significant number of infected computers can be found in the network. The beside table shows top 10 port 135 scanners (consider

TOP 10 Port 135 Scanners	
Count	**Host**
1622940	130.85.111.51
1188912	130.85.81.39
151463	130.85.112.152
127956	130.85.70.96
41586	130.85.66.56
35681	130.85.42.2
28310	130.85.84.224
16759	130.85.153.174
16725	130.85.150.210
15713	130.85.80.224

these machines infected):

As described before, a significant increase of scans for port 2745, 3127, 6129 and 1025 have been observed since the release of IRC bot based worms like agobot. It appears, from the scans results that a significant number of internal computers are infected with one of the variants of this virus. See table “IRC IRC Trojan Alert Correlation table” for a list of infected computers. The remaining ports are probably normal traffic.

OOS Packets

Out of Specification packets are those who violate a certain RFC or implements some new features not yet fully understood by all platforms/OSes, like Explicit Congestion Notification for example. Many scanning tools like NMAP implements some sort of TCP flag combination to observe how a certain machine would respond. Observing the results of valid and invalid flag combinations stimuli, is possible to fingerprint the remote operating system. This kind of attack will result in a series of packets similar to what is contained in the OOS files analyzed. However, some folks are making weird combinations in the way their softwares talks TCP/IP, in order to trick packet filtering devices like firewalls and routers. This becomes very common in the P2P world as a way to get out trough firewalls. If intentional or not (unintentional errors caused by poor code), it generates a lot of noise. The next table shows the flag combinations found it the OOS files:

Count	*Flags*
5147	12****S*
130	12*A**S*
87	********
32	****P***
17	12***R**
3	1*UAPRSF
2	**U*PRSF
2	*2U*PRSF
2	*2UA*RSF
2	12***RSF
1	***A*RSF
1	**U*****
1	**U***SF
1	*2*A*RSF
1	*2U***SF
1	1****RSF
1	1**A*RSF
1	12**P**F
1	12**P*S*
1	12*AP**F
1	12*AP*SF
1	12*APR*F
1	12U***S*
1	12U*P*S*
1	12U*PR**
1	12U*PR*F
1	12U*PRS*
1	12UAP*S*
1	12*A*R**
1	12UAP***

Scan types are classified trough its flags combinations, by using a list taken from SANS track 3 study material:

```
UNKNOWN    -- Ref. spp_portscan.c source code
INVALIDACK -- ACK set, not normal, no SPAU or FULLXMAS
NULL       -- None of SFRPAU
NOACK      -- A flag is missing
FIN        -- F flag
VECNA      -- One of the following: P, U, PU, FP, FU
NMAPID     -- SFPU flags
SPAU       -- SPAU flags
FULLXMAS   -- SFRPAU flags
XMAS       -- FPU flags
SYNFIN     -- SF flags
```

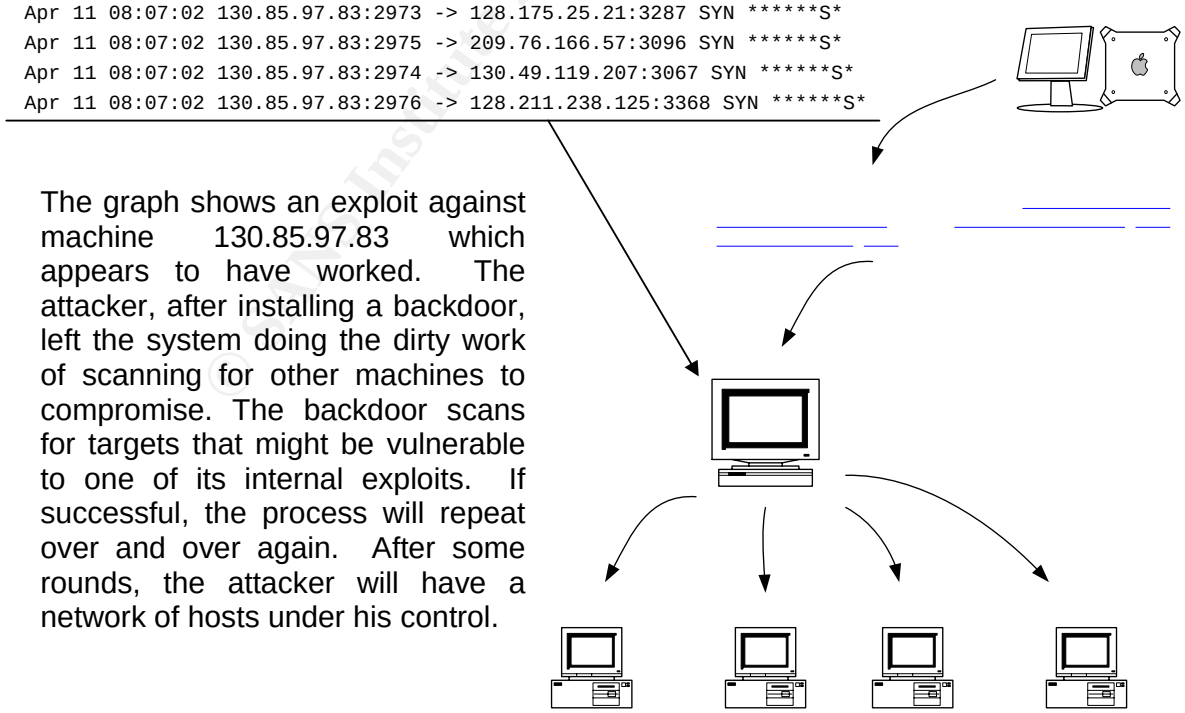
As we can observe from the given table, most alerts are caused by packets with flags 12****S* (ECN) checked. Same can be applied to 12*A**S* and 12***R** (ECN capable three-way-handshake) flag combinations. These packets are likely legitimate packets and not malicious. There are other 87 packets of type NULL SCAN associated to a myriad of external hosts. These packets also correlate with scan and alert logs, so I did a look at contents of some packets dumps and alerts (sample below):

```
04/11-00:19:17.322773 68.121.194.43:6663 -> 130.85.12.4:110 TCP TTL:78 TOS:0x0
ID:4660 IpLen:20 DgmLen:40 ******** Seq: 0xF7E6001  Ack: 0x1D10773  Win: 0x800
TcpLen: 20

04/11-00:41:14.795330 68.121.194.43:6919 -> 130.85.12.4:110 TCP TTL:78 TOS:0x0
ID:4660 IpLen:20 DgmLen:40 ******** Seq: 0xFA88001  Ack: 0x547783B5  Win: 0x800
TcpLen: 20

04/11-00:19:17.322776 [**] Null scan! [**] 68.121.194.43:6663 -> 130.85.12.4:110
04/11-00:41:14.795334 [**] Null scan! [**] 68.121.194.43:6919 -> 130.85.12.4:110
```

They correlate exactly. Most packets come from 68.121.194.43 and are directed to POP3 port of machine 130.85.12.4. Note that IP ID is always the same 4660. This is obviously a result of packet crafting or fragmentation, which would explain the lack of TCP flags as well, but there is no information that indicates these packets are fragments. I suggest take a close look at 130.85.12.4 (mail.umbc.edu), someone are certainly trying to break-in. A brute-force against a POP3 user account is my best guess for now. I will consider the remaining combinations as noise caused by packet corruption, because of its low number of hits.

Link Graphic

```
Apr 11 08:06:59 130.85.97.83:3798 -> 24.162.70.128:2494 UDP
Apr 11 08:06:59 130.85.97.83:2973 -> 128.175.25.21:3287 SYN ******S*
Apr 11 08:06:59 130.85.97.83:2974 -> 130.49.119.207:3067 SYN ******S*
Apr 11 08:06:59 130.85.97.83:2975 -> 209.76.166.57:3096 SYN ******S*
Apr 11 08:06:59 130.85.97.83:2976 -> 128.211.238.125:3368 SYN ******S*
Apr 11 08:06:59 130.85.97.83:2977 -> 65.29.94.227:3124 SYN ******S*
Apr 11 08:06:59 130.85.97.83:2978 -> 24.45.115.10:1433 SYN ******S*
Apr 11 08:07:01 130.85.97.83:2979 -> 65.24.110.243:2212 SYN ******S*
Apr 11 08:07:02 130.85.97.83:2972 -> 130.64.139.62:3720 SYN ******S*
Apr 11 08:07:02 130.85.97.83:2973 -> 128.175.25.21:3287 SYN ******S*
Apr 11 08:07:02 130.85.97.83:2975 -> 209.76.166.57:3096 SYN ******S*
Apr 11 08:07:02 130.85.97.83:2974 -> 130.49.119.207:3067 SYN ******S*
Apr 11 08:07:02 130.85.97.83:2976 -> 128.211.238.125:3368 SYN ******S*
```



The graph shows an exploit against machine 130.85.97.83 which appears to have worked. The attacker, after installing a backdoor, left the system doing the dirty work of scanning for other machines to compromise. The backdoor scans for targets that might be vulnerable to one of its internal exploits. If successful, the process will repeat over and over again. After some rounds, the attacker will have a network of hosts under his control.

External Sources

Below is a list of five external sources of alerts discussed in earlier sections.

216.251.239.251

This host is involved with the NTPDX buffer overflow attack presented in the link graph above.

Trying 216.251.239.251 at ARIN
Trying 216.251.239 at ARIN

OrgName: Navisite, Inc.
OrgID: NAVE
Address: 400 Minuteman Road
City: Andover
StateProv: MA
PostalCode: 01810
Country: US

NetRange: 216.251.224.0 - 216.251.255.255
CIDR: 216.251.224.0/19
NetName: NETBLK-NAVISITE-1
NetHandle: NET-216-251-224-0-1
Parent: NET-216-0-0-0-0
NetType: Direct Allocation
NameServer: MINEDNS001.NAVISITE.NET
NameServer: MINEDNS002.NAVISITE.NET
Comment: ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate: 1999-11-19
Updated: 2002-07-15

TechHandle: ZN103-ARIN
TechName: NaviSite
TechPhone: +1-978-682-8300
TechEmail: arin@navisite.com

OrgTechHandle: MKE2-ARIN
OrgTechName: Kelley, Mike
OrgTechPhone: +1-978-682-8300
OrgTechEmail: mkelley@navisite.com

ARIN WHOIS database, last updated 2004-05-31 19:15
Enter ? for additional hints on searching ARIN's WHOIS database.

68.121.194.43

This host is involved with NULL Scan alerts as discussed in the Portscanning activities section.

whois -h whois.arin.net !net-68-121-194-0-1 ...

CustName: PPPoX Pool - Rback3 SNDG02
Address: 268 Bush St #5000
City: San Francisco
StateProv: CA
PostalCode: 94104
Country: US
RegDate: 2003-09-03
Updated: 2003-09-03

NetRange: 68.121.194.0 - 68.121.195.255
CIDR: 68.121.194.0/23
NetName: SBC068121194000030902
NetHandle: NET-68-121-194-0-1
Parent: NET-68-120-0-0-1
NetType: Reassigned
Comment: For Policy Abuse issues, contact: abuse@swbell.net
Comment: For Technical issues, contact: noc@swbell.net
RegDate: 2003-09-03
Updated: 2003-09-03

TechHandle: PIA2-ORG-ARIN
TechName: IPAdmin-PBI
TechPhone: +1-877-722-3755
TechEmail: IPAdmin-PBI@sbcis.sbc.com

OrgAbuseHandle: APB2-ARIN
OrgAbuseName: Abuse - Pacific Bell
OrgAbusePhone: +1-877-722-3755
OrgAbuseEmail: abuse@pacbell.net

OrgNOCHandle: SPBI-ARIN
OrgNOCName: Support - Pacific Bell Internet
OrgNOCPhone: +1-877-722-3755
OrgNOCEmail: support@pacbell.net

OrgTechHandle: PIA2-ORG-ARIN
OrgTechName: IPAdmin-PBI
OrgTechPhone: +1-877-722-3755
OrgTechEmail: IPAdmin-PBI@sbcis.sbc.com

ARIN WHOIS database, last updated 2004-05-31 19:15
Enter ? for additional hints on searching ARIN's WHOIS database.

128.122.66.204

This is top attacker for alerts regarding IRC bot activities “IRC user /kill”. This is probably the address of IRC server the attacker is using to send commands to the infected hosts. This machine might be also compromised.

Trying 128.122.66.204 at ARIN
Trying 128.122.66 at ARIN

OrgName: New York University
OrgID: NYU
Address: Academic Computing Facility
Address: 251 Mercer Street
City: New York
StateProv: NY
PostalCode: 10012
Country: US

NetRange: 128.122.0.0 - 128.122.255.255
CIDR: 128.122.0.0/16
NetName: NYU-NET
NetHandle: NET-128-122-0-0-1
Parent: NET-128-0-0-0-0
NetType: Direct Assignment
NameServer: CMCL2.NYU.EDU
NameServer: EGRESS.NYU.EDU

NameServer: NYUNSB.NYU.EDU
Comment:
RegDate: 1986-05-02
Updated: 2001-05-21

TechHandle: ZN68-ARIN
TechName: New York University
TechPhone: +1-212-998-3431
TechEmail: NOC@nyu.edu

ARIN WHOIS database, last updated 2004-05-31 19:15
Enter ? for additional hints on searching ARIN's WHOIS databas

nslookup 128.122.66.204
Canonical name: KAPTEREV.ICAS.FAS.NYU.EDU
Addresses:
128.122.66.204

69.140.137.209

This is a machine involved with some alerts of Red Worm and NTPDX attacks.

whois -h whois.geektools.com 69.140.137.209 ...
GeekTools Whois Proxy v5.0.3 Ready.

Checking access for 200.199.37.144... ok.

Final results obtained from whois.arin.net.

Results:
Comcast Cable Communications, Inc. JUMPSTART-3 (NET-69-136-0-0-1)
69.136.0.0 - 69.143.255.255
Comcast Cable Communications, Inc DC15-NROCK1 (NET-69-140-0-0-1)
69.140.0.0 - 69.140.255.255

ARIN WHOIS database, last updated 2004-05-31 19:15
Enter ? for additional hints on searching ARIN's WHOIS database.

whois -h whois.geektools.com !net-69-140-0-0-1 ...
GeekTools Whois Proxy v5.0.3 Ready.
Checking access for 200.199.37.144... ok.
Checking server [whois.arin.net]

Results:
CustName: Comcast Cable Communications, Inc
Address: 3 Executive Campus
Address: 5th Floor
City: Cherry Hill
StateProv: NJ
PostalCode: 08002
Country: US
RegDate: 2004-02-10
Updated: 2004-02-10

NetRange: 69.140.0.0 - 69.140.255.255
CIDR: 69.140.0.0/16
NetName: DC15-NROCK1
NetHandle: NET-69-140-0-0-1
Parent: NET-69-136-0-0-1
NetType: Reassigned
Comment: NONE

RegDate: 2004-02-10

Updated: 2004-02-10

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

OrgTechHandle: IC161-ARIN
OrgTechName: Comcast Cable Communications Inc
OrgTechPhone: +1-856-317-7200
OrgTechEmail: cips_ip-registration@cable.comcast.com

ARIN WHOIS database, last updated 2004-05-31 19:15
Enter ? for additional hints on searching ARIN's WHOIS database.

199.131.21.34

This is the top source host for the “Exploit x86 NOOP” alerts. As we can see for IP owner, this alert is almost 100% sure a false positive.

Trying 199.131.21.34 at ARIN
Trying 199.131.21 at ARIN

OrgName: USDA Office of Operations
OrgID: UOO-2
Address: Suite 133, Building A
Address: 2150 Centre Ave
City: Fort Collins
StateProv: CO
PostalCode: 80526
Country: US

NetRange: 199.128.0.0 - 199.159.255.255
CIDR: 199.128.0.0/11
NetName: USDA-CBLK
NetHandle: NET-199-128-0-0-1
Parent: NET-199-0-0-0-0
NetType: Direct Allocation
NameServer: NS.USDA.GOV
NameServer: NS2.USDA.GOV
NameServer: NS3.USDA.GOV
Comment:
RegDate: 1994-02-08
Updated: 2000-06-16

TechHandle: ZU20-ARIN
TechName: USDA - Office of the ChiefInformation Officer
TechPhone: +1-970-295-5277
TechEmail: Network.Operations@usda.gov

OrgAbuseHandle: ZU20-ARIN
OrgAbuseName: USDA - Office of the ChiefInformation Officer
OrgAbusePhone: +1-970-295-5277
OrgAbuseEmail: Network.Operations@usda.gov

OrgNOCHandle: ZU20-ARIN
OrgNOCName: USDA - Office of the ChiefInformation Officer
OrgNOCPhone: +1-970-295-5277
OrgNOCEmail: Network.Operations@usda.gov

OrgTechHandle: ZU20-ARIN
OrgTechName: USDA - Office of the ChiefInformation Officer

OrgTechPhone: +1-970-295-5277
OrgTechEmail: Network.Operations@usda.gov

ARIN WHOIS database, last updated 2004-05-31 19:15
Enter ? for additional hints on searching ARIN's WHOIS database.

Defensive Recommendations

As explained in the Executive Summary, a lot of good work appears to have been done to protect the security of University's network per comparison with previous analysis reports. Although, there are still many issues to address in order to achieve a more secure environment for users and systems. Good security starts with good prevention, and prevention also means that early warning capabilities are in place to detect attacks in its very beginning. The University Intrusion Detection System is the fundamental piece for this early warning system, but is clear for everyone that look at logs, that those files are extremely full of noise. By noise I mean false positives and informative alerts, which can blind the analysts, so the first defensive recommendation is to improve monitoring capabilities by filtering out non critical alerts, so analysts can focus on what is really important. New versions of Snort have lots of logging improvements such the possibility of logging to distinct files or database depending of alert classification or priority or any other rule you may want to use. For example, see the following configuration extracted from Snort's default configuration file:

```
#You can optionally define new rule types and associate one or more output
#plugins specifically to that type.

#This example will create a type that will log to just tcpdump.
#ruletype suspicious
{
  type log
  output log_tcpdump: suspicious.log
}

#EXAMPLE RULE FOR SUSPICIOUS RULETYPE:
suspicious tcp $HOME_NET any -> $HOME_NET 6667 (msg:"Internal IRC Server";)

#This example will create a rule type that will log to syslog and a mysql
#database:
ruletype redalert
{
  type alert
  output alert_syslog: LOG_AUTH LOG_ALERT
  output database: log, mysql, user=snort dbname=snort host=localhost
}

#EXAMPLE RULE FOR REDALERT RULETYPE:
redalert tcp $HOME_NET any -> $EXTERNAL_NET 31337 \
 (msg:"Someone is being LEET"; flags:A+;)
```

In the above sample two different output types were defined, so alerts can be sent to distinct places for proper analysis. For instance, the output type called log is configured to save alerts in a tcpdump format output, in a file named "suspicious.log", while "redalert" type alerts are configured to be sent to a MySQL database. Similar alternative is recommended as a solution to separate informative alerts, such as 12996 "130.85.30.3

activity" alerts, to separate destination. An alternative for this solution is using the "logto:<filename>" statement inside the rule you may want to save in separate file, however the rule type technique is by far more powerful solution.

Another point to considerate regarding alert output is the reduction of false positives by tuning the detection rules. A key point to consider is filtering out scanning alerts caused by your DNS servers. The majority of scanning alerts were produced by a DNS server, which makes scan files something very difficult to handle. Depending of portscanning plugin being used, ignoring certain hosts is as easy as writing a line similar to

preprocessor portscan-ignorehosts: $DNS_SERVERS

in the snort configuration file.

Avoid rules based on source or destination ports only that don't takes into consideration the session state or flow direction. This may lead to many false positives. Instead, look for more advanced and refined rules to do this job.

At network perimeter, it appears that your firewall rules are too permissive. Allowing outgoing TFTP connections, for example, is not a good idea. Conduct a risk analysis of what is really necessary for users to access in and outbound and block those protocols/connections that represent a risk you don't want to accept.

Viruses, even the old ones, were detected in the network. That mean you are allowing unprotected hosts to connect directly to your network. Updated antivirus at hosts and gateway levels are extremely recommended. We all know how hard is to keep all network machines updated. Things get worse when talking about mobile notebook users. To reduce those problems training and awareness is the first thing to consider. Also, consider the use of "quarantine" network zone, where mobile users are required to connect before connecting to the main network backbone. Inside that quarantine zone, users will have limited access to the network, just enough to update their systems. There are also commercial solutions to accomplish this level of protection. Cisco Systems (www.cisco.com) has recently launched his "Self-defending Networks" initiative, which includes technologies to block non-updated systems to connect to the network.

P2P is a big problem to attack. Disregarding the worm infection threat trough P2P, which is also a high risk; the University must be aware of potential criminal implications of violations of copyrighted material, like mp3 music, found in P2P networks. To avoid potentitial criminal sues, the University's information security manager, should provide "due care" and "due diligence", putting in place all of available resources to help protect against this threat. If you want to count with technical support and constant updates, many content filtering software vendors have P2P protection solutions available. For instance, Websense (www.websense.com) has a solution that watches the network and blocks non-authorized connections, like P2P and Gnutella.

The University's Information Security Policy should include topics about network and computer misuse. This policy must be read and understood by all users, including students. If haven't already done so, develop and publish a comprehensive security policy to drive the University security initiatives.

Appendix – Methodology

Most of commands and procedures used at this work were learned from papers written by Tod Beardsley (http://www.giac.org/practical/Tod_Beardsley_GCIA.doc) and Ian Martin (http://www.giac.org/practical/GCIA/Ian_Martin_GCIA.pdf). I used other students' advice to try SnortSnarf (http://www.silicondefense.com/software/snortsnarf/) for handling of Alerts logs and Ian's techniques to prepare the files before importing it into SnortSnarf. The commands I used, taken from Ian's paper were (thanks Ian):

List of actions to files
$ cat file1 file2 file3 >>file.mrg
All files are cleaned with varients of
$ grep "Jun" scans.mrg >> scans1.mrg
$ grep -v "Jun" scans.mrg >> scans.misc
For the alert files change MY.NET to 130.85 for compatibility with the scans.
$ sd 's/MY.NET/130.85/g' alert.mrg >>alert.mrg1
place the spp_portscan's in a different file and remove from original
$ grep "spp_portscan" alert1.mrg >> alert1.spp
$ grep -v "spp_portscan" alert1.mrg >> alert2.mrg

I then, used Tod's scripts to generate some other useful info I was not getting trough snortsnarf. For analysis of scans and OOS files, I used Ian's commands again to convert files to a more convenient format, then I used Unix commands "cat", "grep", "sed", "awk", "sort" and "uniq' to handle these files. The Ian's commands used at this analysis were (in addition to the commands above):

change "[**]" into ":" for field manipulation
$ sed 's/\[**\]/:/g' alert3.mrg >>alert4.mrg
$ sed 's/\[**\]/:/g' alert1.spp >>alert1.spp1
change "->" into ":" for field manipulation
$ sed 's/\->/:/g' alert5.mrg >> alert6.mrg
Change the MY.NET in the OOS logs
$ sed 's/MY,NET/130.85/g' oos.mrg >>oos1.mrg
Change "->" into ":" for field manipulation
$ sed 's/\->/:/g' scans1.mrg >> scans2.mrg
Data mining was performed using the UNIX commands grep and awk, for example,
awk '$5 == "->" {print $4 ":" $6}' scans1.mrg | cut -d : -f 3 | sort | uniq -c |
sort -rn | less
Or for a list of alerts per IP address
grep "130.85.100.160" alert6.mrg | awk –F : '{print $4}'|sort |uniq –c |sort –rn|less

References

Included in the body of the text.

Last Updated: July 20th, 2017

Upcoming Training

SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS vLive - SEC503: Intrusion Detection In-Depth	SEC503 - 201709,	Sep 11, 2017 - Oct 18, 2017	vLive
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
Baltimore Fall 2017 - SEC503: Intrusion Detection In-Depth	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Scottsdale SEC503	Scottsdale, AZ	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
Community SANS Ottawa SEC503	Ottawa, ON	Oct 16, 2017 - Oct 21, 2017	Community SANS
SANS Berlin 2017	Berlin, Germany	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
San Diego Fall 2017 - SEC503: Intrusion Detection In-Depth	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	vLive
SANS San Diego 2017	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Paris November 2017	Paris, France	Nov 13, 2017 - Nov 18, 2017	Live Event
Community SANS Pensacola SEC503	Pensacola, FL	Nov 27, 2017 - Dec 02, 2017	Community SANS
SIEM & Tactical Analytics Summit & Training	Scottsdale, AZ	Nov 28, 2017 - Dec 05, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS Las Vegas 2018	Las Vegas, NV	Jan 28, 2018 - Feb 02, 2018	Live Event
SANS Dallas 2018	Dallas, TX	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced