Global Information Assurance Certification Paper

Doug Kite
Intrusion Detection in Depth

GCIA Practical Assignment, v3.3
SANS Virginia Beach, July 2002

Table of Contents

Assignment 1 - Describe the State of Intrusion Detection

A package arrives on your front doorstep. It is a small, non-descript box wrapped in plain brown paper. The return address contains no name and is unfamiliar to you. What do you do with the package? What does it contain? Who sent it? When did it arrive? There are many ways you could answer these questions. You could have the package x-rayed or tested by chemical analysis for trace residue. You could have a bomb or drug-sniffing dog inspect the package for you. You could just pick it up and shake it, or drive your car over it to see what happens.

Now picture that millions of these packages arrives at your doorstep every day. Such is the job of the intrusion analyst—millions of packets travel to and from our networks every day. Most of them are innocuous, normal traffic. But how do we know which ones aren't? We obviously cannot look at each one individually, so we need help in determining packets that do not meet certain standards of acceptability. The purpose of this paper is to help the person just starting out in the field by giving an introduction of what intrusion detection systems (IDS) do, a brief history of their development, where they are today, and development of IDS technology illustrated by one tool named Hogwash.

Background

Intrusion detection systems exist to help the analyst examine network traffic. There are many tools that can assist in this endeavor. Perhaps the most basic problem that faces the analyst is seeing the data. While this may sound elementary, the heart of any IDS is some type of engine that "sniffs" the traffic off the wire so that it can be analyzed. One such tool used in the early development

of IDS technology is tcpdump. Capable of running on many different platforms, tcpdump became a popular tool diagnosis and analysis.

Stephen Northcutt at the Naval Surface Warfare Center began writing scripts to help him manage the analysis of network traffic. Shadow was one of the early systems designed to aid with this process. Shadow employs the use of sensors and analysis stations. The sensors gather the data for the analysis machines to process later. One of Shadow's many strengths is its ability to correlate data from many sensors to find coordinated attacks or patterns such as slow scans that stretch out into days or even weeks of time. Huge IDS networks with many distributed sensors have been deployed.

Types of IDS

Shadow is an example of a first generation of IDS—passive systems that log the data and generate alerts on questionable traffic. Another such system is Snort, which was developed by Marty Roesch in November 1998, to monitor the traffic on his home network. Snort is a very flexible system with a powerful signature-based engine.

Snort now boasts a flexible response feature that moves it toward the next generation of IDS, making it capable of actively reacting to unwanted traffic by sending packets to one or both of the hosts involved. In this mode, when Snort sees a packet that matches a rule having the react or resp option, Snort terminates the connection by "forging" a response and sending to the desired host(s). Snort can terminate the offending connection by generating TCP resets or ICMP unreachable traffic. In other words, when Snort sees a connection from host A to host B, it can send a reset to host A that appears to be from host B, and vice versa: send a reset to B that appears to be from A.

Flexible response gives Snort a powerful capability to control traffic flow on the network even though it is primarily a passive device. The main shortcoming of this ability is the ease at which a misconfigured response rule could be disrupting legitimate connections. One of the major challenges of any IDS is dealing with false positives. On a passive IDS, false positives may generate more alerts for the analyst to sift through, but with active response a poorly written rule could have significant impact on network operations.

The next step

A step beyond Snort's flexible response is Hogwash (http://hogwash.sourceforge.net) by Jason Larsen and Jed Haile. Hogwash is categorized as a "packet scrubber" or a signature based firewall and is an example of a reactive IDS. Hogwash is based on and uses the same rule processing engine as Snort. Instead of just alerting on bad traffic, Hogwash will actually drop the packets, not allowing them to enter your network at all. Operating at the link layer of the OSI model, Hogwash allows the creation of a system that protects while remaining transparent itself.

Hogwash uses an in-line topology with two interfaces and all traffic passing through, making it more similar to traditional firewalls than current IDS models.

Since the early 90's the workhorse of security realm has been the firewall. IDS was the passive watcher on the wall, while the firewall actively repels would-be attackers. The problem with this approach is that manual intervention is required causing a delay before information gathered from the IDS was used to make changes on the firewall.

Some systems allow IDSs to add or change access control lists on partner routers or firewalls. This improves response time and eliminates manual intervention, but it also increases the risk that a false positive could trigger an overly restrictive rule. In fact, this can cause a denial of service if a would-be attacker notices this behavior on your network, he can spoof major sites like yahoo.com (or even your potential customers), tricking your system into blocking legitimate traffic from these sites.

Next generation: intrusion prevention

The newest hot buzzword in security today is intrusion prevention. This term refers to a whole class of products designed to proactively mitigate attacks. Anomaly detection and heuristics promise to make intrusion prevention systems (IPSs) smarter and better than even the reactive IDS systems mentioned above.

IPSs promise to take the burden of manual analysis. But anyone who has run an IDS and seen the number of false alarms will have a hard time with the idea of a magical box that just sits there and does it all.

Current IPSs still rely heavily on signatures (even if they automatically update these from the vendor), This means that they are still in a sense reactive, because somebody has to be the first to see an attack, so that a signature to match it can be developed. And the time it takes for the attack to be identified and the vendor to develop and distribute the signature is a window of opportunity for the attackers. The incredibly rapid spread of newly released worms, such as was seen with Code Red and its variants emphasizes the weaknesses of this approach. If a worm is developed that spreads in hours or even minutes, vendors will not have a chance to respond.

Anomaly detection for intrusion detection faces significant challenges. The idea is that a device could monitor "learn" normal patterns and traffic, and alert when data that deviates from the standard is seen. The problem is that network behavior is very hard to pattern statistically. One form of anomaly detection that has shown promise is protocol anomaly detection. Judging whether packets conform to standards such as RFC's, is a much simpler task than finding statistical patterns on the network, but isn't this just another form of comparing the packets to a set of predefined rules (the RFC's)? And what happens when a

certain software vendor decides to extend and embrace some more technology and bend the protocol standards to its favor?

In today's security landscape, the lines blurred between different technologies making the job of comparing competing products difficult. Firewalls have more analysis and alerting capabilities, giving them IDS features. IDSs have capabilities to react and block harmful traffic, making them more like firewalls. All three can have antivirus capabilities, and rely on vendor supplied signatures like antivirus software. Vendors use (and misuse) new terminology to try to differentiate their product from the rest of the pack.

Summary

The diversity of methods of attack will always require different tools and methodologies for protection and mitigation. Firewalls, IDS, and many software utilities serve as tools for the analyst to use in thwarting the bad guys, but no one magic-box solution is soon to change the IDS landscape. The multi-tiered approach of multiple (separate) perimeter defenses, host defenses and sensors remains the best security model today. For the foreseeable future we will continue with a mixed bag of solutions to the age old problem of: what's in all those packets?

References:

Snyder, Joel. "Intrusion Prevention Essentials." SANS Webcast, Dec. 4. 2002. URL: http://www.sans.org/webcasts/december4.php (Dec. 2002)

Messmer, Ellen. "'Intrusion prevention' raises hopes, concerns." Network World, Nov. 4, 2002. URL: http://www.nwfusion.com/news/2002/1104prevention.html (Dec. 2002)

Roesch, Martin and Green, Chris. "Snort Users Manual." URL:http://www.snort.org/docs/writing_rules/ (Dec. 2002)

Larsen, Jason and Haile, Jed. Hogwash Documentation. URL: http://hogwash.sourceforge.net (Dec. 2002)

Briney, Andy. "What Isn't Intrusion Prevention?" Information Security, April 2002. URL: http://www.infosecuritymag.com/2002/apr/note.shtml (Dec. 2002)

Lindstrom, Pete. "Guide to Intrusion Prevention" Information Security, October 2002. URL: http://www.infosecuritymag.com/2002/oct/sidebar.shtml (Dec. 2002)

Schultise, Jeff. GSEC Practical: "Intrusion Prevention as a Logical Evolution from Intrusion Detection". December 2001. URL: http://www.giac.org/practical/jeff_schultise_GSEC.doc (Dec. 2002)

Conz, Jr, James A.GIAC Practical: “Intrusion Prevention: The Next Generation of Security Software?” URL: http://www.giac.org/practical/James_Conz_GCIA.doc (Dec. 2002)

Poor, Mike. GCIA Practical. URL: http://www.giac.org/practical/Mike_Poor_GCIA.doc (Dec. 2002)

Das, Kumar. “Protocol Anomaly Detection for Network-based Intrusion Detection.” January 15, 2002 URL: http://www.sans.org/rr/intrusion/anomaly.php (Dec. 2002)

Assignment 2 - Network Detects

Detect 1 - xdmcp query

1. Source of the trace
http://www.incidents.org/logs/Raw/2002.8.30
Although the name of the file is 2002.8.30, the timestamps on the packets themselves indicate a date of 9/30. As other GCIA practicals using data from this source have noted, this could either be a typo or could be related to other file modifications.

2. Generated by
Snort 1.9.0 (Build 209), using ruleset from snortrules-stable.tar.gz downloaded 12/30/2002.

The alert generated:

```
[**] [1:517:1] MISC xdmcp query [**]
[Classification: Attempted Information Leak] [Priority: 2]
09/30-17:11:03.946507 66.68.128.253:1576 -> 115.74.105.5:177
UDP TTL:110 TOS:0x0 ID:7240 IpLen:20 DgmLen:35
Len: 15
[Xref => arachnids 476]
```

(The alert was repeated 6 times with the only differences being the timestamp and the ID field.)

Packet trace (6 packets):

```
09/30-17:11:03.946507 66.68.128.253:1576 -> 115.74.105.5:177
UDP TTL:110 TOS:0x0 ID:7240 IpLen:20 DgmLen:35
Len: 15
00 01 00 03 00 01 00                                  .......

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

09/30-17:11:05.956507 66.68.128.253:1576 -> 115.74.105.5:177
UDP TTL:110 TOS:0x0 ID:7496 IpLen:20 DgmLen:35
Len: 15
00 01 00 03 00 01 00                                  .......

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

09/30-17:11:09.956507 66.68.128.253:1576 -> 115.74.105.5:177
UDP TTL:110 TOS:0x0 ID:9032 IpLen:20 DgmLen:35
Len: 15
00 01 00 03 00 01 00                                  .......

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

09/30-17:11:17.976507 66.68.128.253:1576 -> 115.74.105.5:177
UDP TTL:110 TOS:0x0 ID:53064 IpLen:20 DgmLen:35
Len: 15
00 01 00 03 00 01 00                                  .......

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

09/30-17:11:33.996507 66.68.128.253:1576 -> 115.74.105.5:177
UDP TTL:110 TOS:0x0 ID:13129 IpLen:20 DgmLen:35
Len: 15
00 01 00 03 00 01 00                                  .......

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

09/30-17:12:06.026507 66.68.128.253:1576 -> 115.74.105.5:177
UDP TTL:110 TOS:0x0 ID:27978 IpLen:20 DgmLen:35
Len: 15
00 01 00 03 00 01 00                                  .......

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+
```

3. Probability the source address was spoofed

It is not very likely that the source address is spoofed. For this type of reconnaissance to be successful a reply needs to be received from the victim, so source spoofing is not probable.

Also, the time difference between the packets indicates that of a normal "doubling-off" of the retransmission backoff time. The time difference between packets starts at 2 seconds, then goes to 4, 8, 16, and finally 32 seconds. This would seem to indicate normal transmission retries.

Checksums on the packets are bad, which could indicate packet crafting. However in this case the bad checksums are probably due to sanitization done to obscure actual ip addresses before posting to www.incidents.org.

4. Description

This is an information gathering attack using the xdmcp protocol. The X display manager (xdm) provides authentication and management for X Windows on many different platforms. xdm allows users to run displays from and control the actions of remote systems. Some systems come with xdm configured by default to allow connections from any host (Caldera, Mandrake, and Solaris). The trace above shows a query from the "attacker" to the "victim" to determine if xdm is accepting connections. If xdm is running and configured to accept connections from any host, it will present a login screen on the attacker's display. On some Linux distributions, xdm will present a list of users that exist on the system as well. The increasing popularity of different distributions of Linux which run xdm and X as their GUI yeilds many potential targets for this attack.

5. Attack mechanism

The query consists of a single UDP packet directed to destination port 177. The payload (hex: 00 01 00 03 00 01 00) is identified by Ethereal as an xdmcp indirect query. The proper way for xdm to respond is to forward the request to other display managers who in turn respond to the inquiring host directly. In practice, xdm usually does not forward the query, responding itself instead.

It is very trivial to query a server in this manner. On a unix machine running X, the user types something like (where x.x.x.x is an ip address):

```
X :2 -query x.x.x.x
```

If the host is vulnerable, the user will be presented with a login screen.

6. Correlations

Though X and xdm have powerful networking capabilities, their weaknesses in the area of security are well known. This particular vulnerability was first reported by Caldera in August 1999 (CSSA-1999:021) and made public in March 2002 by ProCheckUp (http://www.procheckup.com/security_info/vuln_pr0208.html).

Traffic reports at www.dshield.org were searched for the source ip address, but no entries were found. (Dshield.org had only 32 days of data in its online database.) A search for the xdmcp port showed minimal activity for this port in the last month.

CERT Advisory, URL: http://www.kb.cert.org/vuls/id/634847
CVE-2000-0374, URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0374
Bugtraq, URL: http://online.securityfocus.com/bid/1446/

7. Evidence of active targeting

Having no correlating packet data, working only from the alert data it is difficult to determine if active targeting is involved here. Neither of the hosts involved in this trace are found anywhere else in the log file.

Significant changes in the IP ID field on the packets indicate that the sending host was transmitting a high number of packets during this period of time. This would indicate that the activity was related to a much larger scan, even though only one host on the protected net received this data. The value for IP ID is typically incremented by 1 for each packet sent by the host. Thus, active targeting is most likely not involved.

8. Severity

Severity = (criticality + lethality) - (system countermeasures + network countermeasures)
Criticality - 2 because it does not appear to be a server (this is unknown for sure because of limited data). I give it a 2 because this query is looking for unix systems, which would be more powerful and useful to an attacker.
Lethality - 2 information gathering only. Even if the query succeeds, it does not mean the attacker can get in. I give it a 2 because this is a very specific query that if it receives a response would be very useful to an attacker.
Without full knowledge of the network it is difficult to assess countermeasures. Since the IDS saw this packet, there must not be any network countermeasures in place, so we will give that a 1. Since we know nothing of the victim host, we will assume no countermeasures are in place, giving it a 1.

So the severity would be (2 + 2) - (1 + 1) = 2

9. Defensive recommendation

Block UDP port 177 at a border router or firewall. No xdm communications should be allowed to or from hosts on the internet. Better yet, block all unneeded ports at the perimiter, including 177. Do not run X on machines that do not need it. X provides a graphic user interface needed more on workstations than on servers. On hosts that must run xdm, find the file named Xaccess (on Linux with KDE /etc/X11/kdm/Xaccess) in editor and comment out lines that begin with a "*", or replace the "*" with the addresses of hosts that should be allowed. The "*" is a wildcard that allows connections from all hosts. Also, deploy a personal firewall on machines that run X, so that perimeter defense is not the only layer.

10. Test question
What IP protocol does xdmcp use to communicate?
A. TCP
B. UDP
C. ICMP
D. ARP

Answer: B. UDP

This detect was posted to the intrusions list at incidents.org on 12/31/2002. Text above includes changes to the original version I posted based on questions and feedback received. The following questions were asked by Donald Pitts on 01/01/03 11:57PM:

<snip>
>Regarding sections 6 & 7:
>I recommend you consult dshield.org, etc. to try
>and find attacks of the same sort during the
>same time period - those from this source IP and
>others. The results - either way should be included
>in the correlations section and can be used to
>help the conjecture whether this is targeted or
>not.

I had been to dshield.org before, but had not used the reports in depth. Unfortunately, they only have 32 days of data online and this detect was from 9/30/2002. The source address in my detect was not listed in their database for the last 30 days. (It is a RoadRunner address.) And the xdmcp port (177) showed minimal activity on their ports list for the last month. So, though the dshield data did not shed much light on this particular detect, I can see where this is powerful tool for correlation and will certainly use it in the future.

>The IP IDs change pretty significantly between
>some of the packets. Does this add any other
>insight as to what is going on? What else might
>the attacker be doing between retries to our
>victim?

I had noticed this, but didn't follow up on it. After review, this would indicate that the sending host is sending a *LOT* of packets out. This would be evidence that the traffic in the detect was part of a much larger scan.

>8. Severity:
>Criticality - Could the logs be scanned for traffic originating

>from the attacked host to try and infer what role this machine
>plays in the network and therefore facilitate an informed
>guess as its importance?

I had done this, but did not note it in my analysis. There was no other traffic in the logs for either host.

>Lethality - Do we even know if this machine runs Unix or XDMCP?

No, we don't know.

>Countermeasures - What if the IDS is located external to
>network countermeasures? Would you still be able to infer
>that there are no defenses?

No. I guess it is not valid to assume that since the IDS saw the packet, there are no defenses.

>9. Countermeasures:
>Good coverage here. Suggest even blocking all ports not
>needed externally including UDP port 177 at the external
>perimeter. Also consider not running X11 and its associated
>daemons on machines which are not user workstations where
>a direct user interface is required (i.e. servers).
>Another suggestion is to consider running a personal
>firewall on a machine where XDMCP is required so that
>we do not have to depend solely upon the protection
>afforded by X11 to scrutinize these requests.

References:

Benie, Peter. "Communication between XDM and the Chooser." URL: http://www-uxsup.csx.cam.ac.uk/~pjb1008/project/xdm-socket/ (Dec. 2002)
arachNIDS "XDMCP Query" URL: http://www.whitehats.com/info/IDS476 (Dec. 2002)
Security Tracker, URL: http://www.securitytracker.com/alerts/2002/Mar/1003832.html

---

Detect 2 - RPC mountd and pcnfsd query

1. Source of the trace

http://www.incidents.org/logs/Raw/2002.9.23
Although the name of the file is 2002.9.23, the timestamps on the packets themselves indicate a date of 10/23. As other GCIA practicals using data from

this source have noted, this could either be a typo or could be related to other file modifications.

2. Generated by
Snort 1.9.0 (Build 209), using ruleset from snortrules-stable.tar.gz downloaded 12/30/2002.

Sample of the alert generated (8 alerts; 2 shown here):

```
[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
10/23-17:46:45.966507 66.1.161.243:600 -> 32.245.170.117:111
UDP TTL:109 TOS:0x0 ID:42015 IpLen:20 DgmLen:84
Len: 64
[Xref => arachnids 13]
[**] [1:581:2] RPC portmap request pcnfsd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
10/23-17:46:50.776507 66.1.161.243:600 -> 32.245.170.117:111
UDP TTL:109 TOS:0x0 ID:42527 IpLen:20 DgmLen:84
Len: 64
[Xref => arachnids 22]
```

Rule(s) that generated the alerts:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request
mountd"; content:"|01 86 A5 00 00|";offset:40;depth:8;
reference:arachnids,13; classtype:rpc-portmap-decode; sid:579; rev:2;)
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request
pcnfsd"; content:"|02 49 f1 00 00|";offset:40;depth:8;
reference:arachnids,22; classtype:rpc-portmap-decode; sid:581; rev:2;)
```

Packet trace (8 packets):

```
10/23-17:46:45.956507 66.1.161.243:600 -> 32.245.170.117:111
UDP TTL:109 TOS:0x0 ID:41759 IpLen:20 DgmLen:84
Len: 64
FF 05 86 2F 00 00 00 00 00 00 00 02 00 01 86 A0  .../............
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 01 86 A5 00 00 00 01  ................
00 00 00 11 00 00 00 00                          ........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

10/23-17:46:45.966507 66.1.161.243:600 -> 32.245.170.117:111
UDP TTL:109 TOS:0x0 ID:42015 IpLen:20 DgmLen:84
Len: 64
FF 05 86 2F 00 00 00 00 00 00 00 02 00 01 86 A0  .../............
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 01 86 A5 00 00 00 01  ................
00 00 00 11 00 00 00 00                          ........
```

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

10/23-17:46:50.776507 66.1.161.243:600 -> 32.245.170.117:111
UDP TTL:109 TOS:0x0 ID:42527 IpLen:20 DgmLen:84
Len: 64
FF 05 86 30 00 00 00 00 00 00 00 02 00 01 86 A0  ...0............
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 02 49 F1 00 00 00 02  ..........I.....
00 00 00 11 00 00 00 00                          ........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

10/23-17:46:50.926507 66.1.161.243:600 -> 32.245.170.117:111
UDP TTL:109 TOS:0x0 ID:42783 IpLen:20 DgmLen:84
Len: 64
FF 05 86 30 00 00 00 00 00 00 00 02 00 01 86 A0  ...0............
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 02 49 F1 00 00 00 02  ..........I.....
00 00 00 11 00 00 00 00                          ........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

10/23-17:46:59.046507 66.1.161.243:600 -> 32.245.170.117:111
UDP TTL:109 TOS:0x0 ID:44831 IpLen:20 DgmLen:84
Len: 64
FF 05 86 31 00 00 00 00 00 00 00 02 00 01 86 A0  ...1............
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 01 86 A5 00 00 00 01  ................
00 00 00 11 00 00 00 00                          ........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

10/23-17:46:59.226507 66.1.161.243:600 -> 32.245.170.117:111
UDP TTL:109 TOS:0x0 ID:45087 IpLen:20 DgmLen:84
Len: 64
FF 05 86 31 00 00 00 00 00 00 00 02 00 01 86 A0  ...1............
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 01 86 A5 00 00 00 01  ................
00 00 00 11 00 00 00 00                          ........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+

10/23-17:47:03.756507 66.1.161.243:600 -> 32.245.170.117:111
UDP TTL:109 TOS:0x0 ID:45343 IpLen:20 DgmLen:84
Len: 64
FF 05 86 32 00 00 00 00 00 00 00 02 00 01 86 A0  ...2............
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 02 49 F1 00 00 00 02  ..........I.....
00 00 00 11 00 00 00 00                          ........
```

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
+=+

10/23-17:47:04.116507 66.1.161.243:600 -> 32.245.170.117:111
UDP TTL:109 TOS:0x0 ID:45599 IpLen:20 DgmLen:84
Len: 64
FF 05 86 32 00 00 00 00 00 00 00 02 00 01 86 A0  ...2............
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 02 49 F1 00 00 00 02  ..........I.....
00 00 00 11 00 00 00 00                          ........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
+=+
```

3. Probability the source address was spoofed

Even though this probe involves UDP traffic which is easily spoofed, the likelihood that spoofing is involved here is low. Because of the nature of this query, in order to be successful the attacker needs to see the result that is returned from the remote host.

4. Description

This is a probe for two specific rpc services: mountd and pcnfsd. Sun RPC services are used for file and printer sharing between hosts. Mountd and pcnfsd are services which run on Unix hosts that allow other hosts and pc's to access files (pcnfsd allows access to filesystems without having to worry about UID/GID mappings). The portmapper service runs on port 111 and responds to queries identifying what ports other services are running on. So the example traffic cited here is an attacker querying a host to see if these services are running, and if so, on what ports. If a vulnerable version is found, then the attacker could access files on the remote host or perhaps gain root access.

5. Attack mechanism

The probe involves UDP traffic to destination port 111. In the 8 packets captured, there are 2 attempts to query for mountd and 2 for pcnfsd. Each of the 4 attempts consists of 2 packets with the same RPC transaction ID (XID). The source port for all the packets is 600 (a privileged port). Specifically, these are rpc getport queries, which means the attacker is querying the host, asking what port mountd and pcnfsd are listening on.

There are quite a few known exploits for rpc services. Since the probe is looking for two specific services, mountd and pcnfsd, it is safe to assume that the attacker has exploit(s) in hand and is looking for a place to use them. If a response is received to this query the attacker knows what ports the vulnerable services are listening on.

6. Correlations

This is a widely seen scan. In fact, it relates to the top vulnerability listed on the

SANS/FBI Top 20 Vulnerabilities list for Unix systems (http://www.sans.org/top20/#U1).

Brian Speegle noted a similar scan as far back as Oct. 2000, (http://www.sans.org/y2k/102400.htm).

There are several known exploits for rpc services. John Vajda lists several known vulnerabilities in his practical dated June 15, 2000, (http://www.giac.org/practical/Joh_Vajda.doc).

7. Evidence of active targeting

Since there is no traffic from the source host to any other hosts on the destination net, active targeting is probable. Without correlating data from previous days it is hard to tell. Perhaps this is a follow-up of a prior scan or other probe.

8. Severity

Severity = (criticality + lethality) - (system countermeasures + network countermeasures)
Criticality - 2 There is no evidence that the victim is a server, but these services normally run on Unix workstations.
Lethality - 1 This is an informational scan only.
With the limited data from the packet log it is difficult to assess countermeasures. Since the IDS saw this packet, there must not be any nework countermeasures, so we will give that a 1. Since we know nothing of the victim host, we will assume no countermeasures are in place, giving it a 1.

So the severity would be (2 + 1) - (1 + 1) = 1

9. Defensive recommendation

Disable rpc services on all hosts where they are not necessary. These services come enabled by default on most Unix and Linux systems. Block port 111 (TCP and UDP) on a border router or firewall. You also need to block ports used by the rpc services themselves, in the range of 32770-32789 (TCP and UDP). Install patches on systems that must run rpc services.

For a more proactive approach, you may wish to actively scan your network for these servcies, to make sure that new machines are not brought online with rpc enabled. There are many tools that could be used for this purpose: two examples are Nessus (www.nessus.org) or nmap (www.insecure.org/nmap/).

10. Test question

It is easier to spoof a source IP address with UDP traffic. This is because:
A. There is no IP ID field.
B. The type of service field (TOS) is always 0.
C. It is a "connectionless" protocol.
D. There are no options for window size.

Answer: C - UDP is a "connectionless" protocol. There is no three-way handshake as there is with TCP, making it easier to spoof.

References:
arachNIDS. "IDS 13 Portmap-request-mountd". URL: http://www.whitehats.com/info/IDS13
arachNIDS. "IDS 22 Portmap-request-pcnfsd". URL: http://www.whitehats.com/info/IDS22
Mitre. "CVE CAN-1999-0632". URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0632
SecurityFocus. "SGI Security Advisory". URL: http://online.securityfocus.com/advisories/4333 (Aug. 2002)
CERT Advisory CA-1996-08. "Vulnerabilites in PCNFSD". URL: http://www.cert.org/advisories/CA-1996-08.html
Reece, David. "Information Security Paper: Rpcbind and Portmapper". URL: http://www.sans.org/newlook/resources/IDFAQ/blocking.htm

---

Detect 3 - Tiny Fragments

1. Source of the trace

http://www.incidents.org/logs/Raw/2002.10.16
Although the name of the file is 2002.10.16, the timestamps on the packets themselves indicate a date of 11/16. As other GCIA practicals using data from this source have noted, this could either be a typo or could be related to other file modifications.

2. Generated by

Snort 1.9.0 (Build 209), using ruleset from snortrules-stable.tar.gz downloaded 12/30/2002.

Sample of the alert generated:

```
[**] [1:522:1] MISC Tiny Fragments [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/16-06:42:06.146507 66.81.110.134 -> 170.129.139.231
TCP TTL:231 TOS:0x0 ID:0 IpLen:20 DgmLen:40 MF
Frag Offset: 0x0800   Frag Size: 0xFFFFF814
```

Rule that generated the alert:

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC Tiny Fragments";
fragbits:M; dsize: < 25; classtype:bad-unknown; sid:522; rev:1;)
```

Packet dump (snort output):

```
11/16-06:42:06.146507 66.81.110.134 -> 170.129.139.231
TCP TTL:231 TOS:0x0 ID:0 IpLen:20 DgmLen:40 MF
Frag Offset: 0x0800   Frag Size: 0xFFFFF814
```

```
81 17 00 50 03 A8 07 0C 03 A8 07 0C 50 04 00 00  ...P........P...
31 D1 00 00                                      1...
```

3. Probability the source address was spoofed
Based on the conclusions below, including the evidence of packet crafting, the odds are low that the source address is spoofed. If this is an attempt to gather information about the destination host, then a reply would be needed.

4. Description of the attack
This appears to be an stealthy attempt to elicit information about a host. This conclusion is based on the following evidence:

Fragmentation is rare on most networks today. This is because higher level protocols of the TCP/IP stack handle address this. It is common for hosts to set the don't fragment (DF) bit on packets. If the don't fragment bit is set and the packet is too large for the MTU of a router, the router should send back in ICMP message to the host informing it of an acceptable MTU. The host then resends the data with a smaller packet size, so the receiving host never sees any fragments.

A common use for fragmentation today is avoiding firewalls and IDSes. A firewall that is filtering on protocol header information, such as tcp or udp port numbers may allow fragmented packets to pass. This is because header information would only be included in the *first* packet, there is no header information in following fragments. It is also possible for the sub-protocol (tcp, udp, icmp) header itself to be fragmented. If the firewall is not re-assembling the fragments, it may allow the traffic--but the receiving host reassembles the packet and the payload is delivered.

Programs such as fragrouter (by Dug Song, http://www.anzen.com/research/nidsbench/fragrouter.html) fragment data for purposes of evasion.

It would appear that this trace is not such an attempt, because it involves only one packet, so what is the purpose of this packet?

5. Attack mechanism
The small size of the packet payload is what triggered the Snort alert. The rule specifies to alert if the payload size is less than 25 and the and the "more fragments follow" bit is set. The logic behind the rule seems to be: why would a host send only 25 bytes of data if there is more data to follow? The most efficient way would be to send as much data as possible if there is more data. This indicates that the packet was crafted.

The IP ID (and thus the fragment ID) are 0. While documentation suggests that this is not very common, I found many such examples from data collected over a month on my own network.

Is the packet corrupt? Probably not because the checksum is correct that the IP length field is correct. And if it were a "mis-fire", sent to a mistaken destination address, you would expect to see more than one packet.

The packet does not appear to be damaged, yet it is not a normal fragment, and it is the only fragment in the stream. This leads to the conclusion that this is some type of stealthy network mapping.

So why send such a packet? When an IP fragment is sent to a host that does not exist, the router should reply with an ICMP unreachable. This fits with an inverse mapping technique, where the attacker sends these packets, and receives replies on hosts that *don't* exist. So the packets sent for which no reply is received are possible "live" targets.

Why only one packet? Perhaps the intent here is a "low and slow" type scan to avoid attention or alerts.

Another possible explanation of this packet is some type of covert communication. Since there is a small payload, a message or command could be included. If the listening host is expecting such a packet, then this may be a way to pass data through the firewall. This is probably not the case, because since there is no tcp port field in the packet, there could not be a "listener" on the receiving host. The remote host would have to have it's nic in promiscuous mode, sniffing for such traffic in order to receive it. While not impossible, this makes this type of communication more difficult and less likely in this case. Also, there is no other activity from the receiving host. One would expect that if a command were sent, some action would be taken. It is possible that a command could be sent that does not trigger until days or weeks later. This is typical of some communication between master and slave (drones) systems that have been trojaned. I do not believe such communication is the case in this example.

6. Correlations:
There was no other traffic in the log file involving either host from this packet.

No traffic from the source host was found on dshield.org during the last month.

The class notes from the SANS Intrusion Detection Track contain similar patterns. The IDS Signatures and Analysis notes, in the section titled "Network Mapping/Information Gathering", page 10-44 discusses using fragmented IP datagrams without a zero offset for mapping purposes.

Other information on stealthy probes and scans:
"Practical Automated Detection of Stealthy Portscans." by Stuart Staniford, James A. Hoagland and Joseph M. McAllerney.
URL: http://www.silicondefense.com/pptntext/Spice-JCS.pdf

Julie Lefebvre noted similar traffic (also at a low rate) in her practical detect #4, dated Feb. 8, 2000.
(http://www.giac.org/practical/Julie_lefebvre.doc)

7. Evidence of active targeting
None. This is probably part of a larger scan. It may be a slow scan in order to avoid detection.

8. Severity
Severity = (criticality + lethality) - (system countermeasures + network countermeasures)
Criticality - 1 There is no evidence that the victim is a server.
Lethality - 1 This is an informational scan only.
With the limited data from the packet log it is difficult to assess countermeasures. Judging by other traffic in this file, there must not be any nework countermeasures, so we will give that a 1. Since we know nothing of the victim host, we will assume no countermeasures are in place, giving it a 1.

So the severity would be (1 + 1) - (1 + 1) = 0

9. Defensive Recommendations
Add 66.81.110.134 to a watch list. Search for for activity from this host in other day's logs. (The notes on sanitization of data from incidents.org/logs/Raw, where this trace was obtained, said that ip addresses had been changed consistently for all data from one day, but not necessarily from one day to the next. In other words, I take it that the ip address may be changed to one thing on one day, but something different in the next day's log. For this reason, I did not try to correlate and search for other instances from the same source address in other days' data.) If other scan activity is detected from this host, add it to a block list at the perimeter.

10. Test question
What is the normal response of a router when it receives a fragmented packed addressed to a host that does not exist?
A. silently drop the packet
B. send an ICMP message back to the source host
C. add the sending host to a drop list
D. keel over and die

Answer: B

Assignment 3 - Analyze This

Executive Summary

Intrusion detection systems produce large amounts of data. The challenge for the analyst is to properly summarize the data into usable form, and extract the important information that requires attention. The purpose of this study is to analyze five days of data, from December 17-21, 2002. Three types of data were included: alerts, scan logs, and out-of-spec files. Alerts generated by the IDS can be false positives, real but harmless, or inappropriate traffic. Our hope is to summarize the massive amount of data and give insight about the dangerous traffic found.

Several internal hosts included in this study are obviously comprised and require immediate action. Suggestions will be made on mitigating the damage and improving IDS and network performance.

Some traffic (such as file sharing software) may be benign yet still an undesirable effect on network performance. The organization's acceptable use policy should be checked to determine if such traffic should be blocked.

Files Analyzed

Alerts	Scans	Out of Spec
alert.021217	scans.021217	OOS_Report_2002_12_18_22751
alert.021218	scans.021218	OOS_Report_2002_12_19_25142
alert.021219	scans.021219	OOS_Report_2002_12_20_27421
alert.021220	scans.021220	OOS_Report_2002_12_21_31218
alert.021221	scans.021221	OOS_Report_2002_12_22_1703

(Note: The OOS files contain the prior day's data, i.e. the file named 2002_12_18 actually contains entries for the day of 12/17.)

Analysis

In the data selected for analysis, there were a total of about 717,199 individual alerts. (I say "about" because there were some lines that were not properly formatted: 103 lines were truncated entries and 62 lines involved two entries on the same line run together.) There were 47 different signatures represented (including portscans).

Following is a list of alert types by count:

```
45877  SMB Name Wildcard
45564  spp_http_decode: IIS Unicode attack detected
22433  TFTP - External UDP connection to internal tftp server
17788  PORTSCAN DETECTED
12043  Watchlist 000220 IL-ISDNNET-990517
```

```
9633  High port 65535 tcp - possible Red Worm - traffic
8331  spp_http_decode: CGI Null Byte attack detected
7374  Incomplete Packet Fragments Discarded
4714  High port 65535 udp - possible Red Worm - traffic
2211  Port 55850 tcp - Possible myserver activity - ref. 010313-1
2175  Watchlist 000222 NET-NCFC
1486  IDS552/web-iis_IIS ISAPI Overflow ida nosize
1403  Queso fingerprint
1244  External RPC call
 975  IRC evil - running XDCC
 737  EXPLOIT x86 NOOP
 446  Tiny Fragments - Possible Hostile Activity
 254  SMB C access
 218  SUNRPC highport access!
 195  Port 55850 udp - Possible myserver activity - ref. 010313-1
 144  Null scan!
 125  NMAP TCP ping!
  72  TFTP - Internal UDP connection to external tftp server
  64  TCP SRC and DST outside network
  62  Attempted Sun RPC high port access
  53  EXPLOIT x86 setuid 0
  42  Possible trojan server activity
  31  EXPLOIT x86 setgid 0
  16  DDOS mstream client to handler
  15  RFB - Possible WinVNC - 010708-1
  15  EXPLOIT x86 stealth noop
  12  External FTP to HelpDesk MY.NET.70.50
  10  ICMP SRC and DST outside network
   7  External FTP to HelpDesk MY.NET.70.49
   4  EXPLOIT NTPDX buffer overflow
   3  FTP DoS ftpd globbing
   3  DDOS shaft client to handler
   3  Bugbear@MM virus in SMTP
   1  TFTP - Internal TCP connection to external tftp server
   1  SYN-FIN scan!
   1  SNMP public access
   1  NIMDA - Attempt to execute cmd from campus host
   1  MY.NET.30.3 activity
   1  Back Orifice
```

The first area of concern should be to identify which of these alerts involved traffic outgoing from the protected network. While incoming alerts are important and tell us what types of attacks our systems are facing, many of these alerts, though not false alarms, can be quickly dismissed. For example, a server may receive thousands of unicode attack alerts, but if it is not running IIS (or any web server at all) these alerts may be safely ignored. Outgoing alerts, on the other hand, can only be one of three things:

1. A compromised host that is being remotely controlled or executing worm code
2. A user on the internal net is actively attacking or scanning hosts
3. A false positive

By examining outgoing alerts, we can quickly determine which hosts on the internal net may have already been compromised. Following is a list of alerts that have MY.NET.x.x as the source address, indicating questionable traffic leaving the internal network:

Alerts with MY.NET as the source network:

```
count  Type
42582  spp_http_decode: IIS Unicode attack detected
22436  TFTP - External UDP connection to internal tftp server
 8291  spp_http_decode: CGI Null Byte attack detected
 6220  Incomplete Packet Fragments Discarded
 5552  High port 65535 tcp - possible Red Worm - traffic
 2334  High port 65535 udp - possible Red Worm - traffic
 1246  Port 55850 tcp - Possible myserver activity - ref. 010313-1
  976  IRC evil - running XDCC
  189  Port 55850 udp - Possible myserver activity - ref. 010313-1
   67  TFTP - Internal UDP connection to external tftp server
   24  Possible trojan server activity
    8  RFB - Possible WinVNC - 010708-1
    1  Watchlist 000222 NET-NCFC
    1  NIMDA - Attempt to execute cmd from campus host
    1  MY.NET.30.3 activity
```

Outgoing IIS Unicode attacks

The Snort http_decode preprocessor can be known to give false positives, but probably not all of these 42,582 alerts are such. As the Snort FAQ points out (http://www.snort.org/docs/faq.html#4.12) certain encoding in cookies and SSL can cause false unicode alerts. This can be turned off by using the -unicode switch on the line for spp_http_decode in snort.conf. Still, the number of alerts would indicate at least the possibility that internal hosts have been compromised by Nimda, Code Red, Sadmind or other common attacks that use unicode to "hide" a directory traversal or other attack and are now scanning for other victims. (These and other worm exploits use every compromised host to begin scanning and attacking other hosts.) The best way to determine this would be to look at the packet log for these alerts. Unfortunately, the packet logs are not available with this data set, so further analysis is challenging. Since these worms typically involve large scans, I looked at the top three hosts involved in outgoing IIS unicode alerts.

MY.NET.84.169 topped the list at 8949 outgoing unicode alerts, 83% of the total alerts for this host. All of the other alerts from the host related to outgoing portscans. The scans files do not use the 'MY.NET' substitution on ip addresses, so correlation between alert and scan logs was uncertain (there were entries for 65.96.84.169). This host should be quarantined until further investigation can be done. If it is running IIS, the latest service packs should be installed.

MY.NET.85.74 was next with 3585 outgoing unicode alerts. All of the alerts seen for this host are of this type. Judging by the timing of the alerts and the fact that

all alerts within a few seconds of each other are to the same host, I would categorize these as false alarms. Example partial alerts follow:

```
12/17-09:15:32.222580 MY.NET.85.74:1114 -> 64.12.42.116:80
12/17-09:15:32.397558 MY.NET.85.74:1116 -> 64.12.42.116:80
12/17-09:15:33.111262 MY.NET.85.74:1119 -> 64.12.42.116:80
12/17-09:15:54.011099 MY.NET.85.74:1125 -> 207.200.89.193:80
12/17-09:15:55.625834 MY.NET.85.74:1125 -> 207.200.89.193:80
12/17-09:15:56.315342 MY.NET.85.74:1126 -> 207.200.89.193:80
12/17-09:15:58.460831 MY.NET.85.74:1130 -> 207.200.89.193:80
12/17-09:15:58.565643 MY.NET.85.74:1130 -> 207.200.89.193:80
12/17-09:38:36.949272 MY.NET.85.74:1254 -> 64.12.151.211:80
12/17-09:38:50.458597 MY.NET.85.74:1260 -> 207.200.89.193:80
12/17-09:38:55.103571 MY.NET.85.74:1260 -> 207.200.89.193:80
12/17-11:51:37.735163 MY.NET.85.74:3177 -> 195.93.80.120:80
12/17-11:51:38.110732 MY.NET.85.74:3175 -> 195.93.80.120:80
12/17-11:51:38.930940 MY.NET.85.74:3175 -> 195.93.80.120:80
12/17-17:04:59.240432 MY.NET.85.74:3272 -> 64.12.151.211:80
12/17-17:05:38.391280 MY.NET.85.74:3293 -> 207.200.89.196:80
12/18-10:16:19.494252 MY.NET.85.74:1034 -> 207.200.86.66:80
12/18-10:16:25.735278 MY.NET.85.74:1054 -> 207.200.86.66:80
12/18-10:16:26.287337 MY.NET.85.74:1062 -> 207.200.86.66:80
12/18-10:16:27.329461 MY.NET.85.74:1064 -> 207.200.86.66:80
12/18-10:16:27.582255 MY.NET.85.74:1066 -> 207.200.86.66:80
```

If this were a scan, you would see many different destination hosts within a few seconds of each other. What we see here is entries that are within seconds of each other refer to the same host. The same number of packets is not sent to each host, and the source port numbers seem to increase normally.

MY.NET.122.118 was next with 2578 outgoing unicode alerts. The other alerts seen from this host were CGI Null Byte attack alerts. Since these alerts are both related to the http_decode preprocessor (Snort FAQ 4.12), these are probably false positives as well.

TFTP - External UDP connection to internal tftp server (22,436 alerts)
Sample log data (partial alert info):

```
12/17-00:10:51.825473 MY.NET.111.235:69 -> 192.168.0.253:6322
12/17-00:21:27.560581 MY.NET.111.235:69 -> 192.168.0.253:9163
```

This appears to be a custom rule. Most of these alerts deal with a destination address of 192.168.0.253. 192.168 is a reserved network that should not be seen on the internet. However, it is possible for a misconfigured device to send data out with the private source address as long as the destination address is valid. If the border router is not doing egress filtering, the packet may be delivered. Since the source port on the packets is 69, it indicates a response from the tftp server to the client (high-numbered) port. So it is possible that a misconfigured device on either the same net as the IDS or on the internet triggered these alerts, and the alerts are on the responses to these packets. If the 192.168 address space is used on the internal network, perhaps some packets are not having addresses

translated properly. The fact that this is UDP traffic lends itself to this type of problem.

The message on the rule says "External UDP connection to internal tftp server". Since these are clearly outgoing packets, I would say that this rule needs to be reviewed and possibly tightened up.

Possible Trojan server activity (24 alerts)
These alerts were probably triggered by a rule looking for activity on port 27374, a known trojan port (a presumption because of lack of access to the ruleset that generated the alerts). However, port 27374 can be used as a valid client port in normal activity. In the following sample, the responses that come from source port 21 (ftp), 80 (www), 6346 (gnutella) and 1214 (Kazaa) are probably false positives (again, it is impossible to know for sure without full packet logs).

Sample alert data (partial info):

```
12/17-09:56:47.456486 MY.NET.162.91:21 -> 218.104.218.2:27374
12/17-13:16:45.807111 MY.NET.86.102:6346 -> 80.135.90.167:27374
12/17-13:16:46.520055 MY.NET.86.102:6346 -> 80.135.90.167:27374
12/17-13:16:47.414428 MY.NET.86.102:6346 -> 80.135.90.167:27374
12/17-13:16:50.456258 MY.NET.86.102:6346 -> 80.135.90.167:27374
12/19-15:35:20.716372 MY.NET.157.52:80 -> 204.193.75.8:27374
12/19-15:35:20.716930 MY.NET.157.52:80 -> 204.193.75.8:27374
12/19-15:35:20.716960 MY.NET.157.52:80 -> 204.193.75.8:27374
12/19-15:35:20.727555 MY.NET.157.52:80 -> 204.193.75.8:27374
12/19-15:35:20.727820 MY.NET.157.52:80 -> 204.193.75.8:27374
12/19-15:35:20.755853 MY.NET.157.52:80 -> 204.193.75.8:27374
12/19-15:35:20.756595 MY.NET.157.52:80 -> 204.193.75.8:27374
12/20-03:36:54.773531 MY.NET.113.4:1214 -> 217.81.85.194:27374
12/20-03:36:54.982821 MY.NET.113.4:1214 -> 217.81.85.194:27374
12/20-03:36:54.983051 MY.NET.113.4:1214 -> 217.81.85.194:27374
12/20-03:36:55.135676 MY.NET.113.4:1214 -> 217.81.85.194:27374
12/21-04:31:57.503241 MY.NET.113.4:1214 -> 63.225.96.164:27374
12/21-11:18:11.600566 MY.NET.113.4:1214 -> 12.252.62.103:27374
12/21-11:18:11.821728 MY.NET.113.4:1214 -> 12.252.62.103:27374
12/21-11:18:11.861090 MY.NET.113.4:1214 -> 12.252.62.103:27374
12/21-11:18:12.091584 MY.NET.113.4:1214 -> 12.252.62.103:27374
12/21-14:15:53.200911 MY.NET.140.47:27374 -> 62.123.117.82:7777
12/21-14:16:02.373726 MY.NET.140.47:27374 -> 62.123.117.82:7777
12/21-14:16:57.398963 MY.NET.140.47:27374 -> 62.123.117.82:7777
```

Since a trojaned machine would be listening on port 27374 and responding with that as its source port, the most interesting entry above is the one from MY.NET.140.47, especially considering that the destination port (7777) is another known trojan port. This machine does appear to have been trojaned and should be quarantined. This was confirmed by searching alerts for other entries involving MY.NET.140.47, which revealed many portscans originating from this host, and the following:

```
12/17-00:07:32.281464  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] 131.164.148.164:8875 -> MY.NET.140.47:55850
```

```
12/17-03:16:33.805644  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.140.47:65535 -> 24.157.102.178:8888
12/17-03:16:35.143845  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.140.47:65535 -> 24.157.102.178:8888
12/17-03:16:40.605454  [**] SUNRPC highport access! [**]
62.30.48.69:8888 -> MY.NET.140.47:32771
12/18-02:29:37.271455  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] MY.NET.140.47:55850 -> 131.164.148.164:8875
12/18-02:29:37.425883  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] 131.164.148.164:8875 -> MY.NET.140.47:55850
12/18-09:48:06.269856  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.140.47:65535 -> 80.181.141.252:7777
12/18-09:48:12.474231  [**] SUNRPC highport access! [**]
65.35.109.221:3333 -> MY.NET.140.47:32771
12/20-22:26:01.667209  [**] DDOS shaft client to handler [**]
64.230.128.142:3456 -> MY.NET.140.47:20432
12/20-22:26:06.560561  [**] DDOS shaft client to handler [**]
64.230.128.142:3456 -> MY.NET.140.47:20432
12/20-22:26:06.638685  [**] DDOS shaft client to handler [**]
64.230.128.142:3456 -> MY.NET.140.47:20432
12/21-14:15:53.200911  [**] Possible trojan server activity [**]
MY.NET.140.47:27374 -> 62.123.117.82:7777
12/21-14:16:02.373726  [**] Possible trojan server activity [**]
MY.NET.140.47:27374 -> 62.123.117.82:7777
12/21-14:16:57.398963  [**] Possible trojan server activity [**]
MY.NET.140.47:27374 -> 62.123.117.82:7777
```

These entries show several suspicious connections, some involving other known trojan ports (7777 and 3456). Combined with the portscan activity from this host, it is clear that it has been compromised with a trojan (perhaps more than one) and is being controlled remotely.

Further investigation shows that another local host had contact with 64.230.128.142 (and on the same trojan port 3456):

```
12/21-11:17:08.886370  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] MY.NET.140.136:55850 -> 64.230.128.142:3456
12/21-11:17:08.946128  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] 64.230.128.142:3456 -> MY.NET.140.136:55850
```

The scan log shows that prior to the above contact on port 3456, MY.NET.140.136 had been scanning for this port:

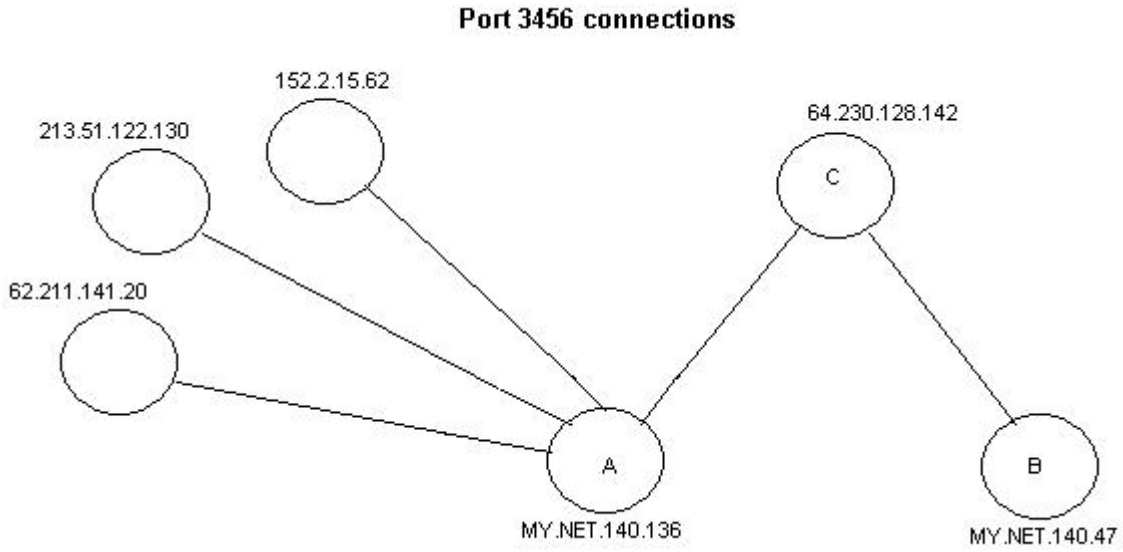
```
Dec 19 09:17:06 MY.NET.140.136:60413 -> 64.230.128.142:3456 SYN
******S*
Dec 19 09:17:31 MY.NET.140.136:60492 -> 64.230.128.142:3456 SYN
******S*
Dec 19 09:24:12 MY.NET.140.136:61254 -> 64.230.128.142:3456 SYN
******S*
Dec 19 09:26:06 MY.NET.140.136:61501 -> 64.230.128.142:3456 SYN
******S*
Dec 21 08:05:27 MY.NET.140.136:62765 -> 64.230.128.142:3456 SYN
******S*
Dec 21 08:05:39 MY.NET.140.136:62775 -> 64.230.128.142:3456 SYN
```

```
******S*
Dec 21 17:10:58 MY.NET.140.136:58820 -> 64.230.128.142:3456 SYN
******S*
```

In addition, MY.NET.140.136 had contact with other hosts via port 3456:

```
12/20-15:19:43.734702  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] MY.NET.140.136:55850 -> 213.51.122.130:3456
12/20-15:19:48.747548  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] 213.51.122.130:3456 -> MY.NET.140.136:55850
12/20-15:19:48.764158  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] MY.NET.140.136:55850 -> 213.51.122.130:3456
12/20-15:19:48.765436  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] MY.NET.140.136:55850 -> 213.51.122.130:3456
12/20-18:54:28.349719  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] MY.NET.140.136:55850 -> 62.211.141.20:3456
12/20-18:54:33.001984  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] MY.NET.140.136:55850 -> 62.211.141.20:3456
12/20-18:54:49.688822  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] MY.NET.140.136:55850 -> 62.211.141.20:3456
12/20-18:55:11.926238  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] MY.NET.140.136:55850 -> 62.211.141.20:3456
12/20-18:55:41.686437  [**] Port 55850 tcp - Possible myserver activity
- ref. 010313-1 [**] MY.NET.140.136:55850 -> 62.211.141.20:3456
12/21-09:06:03.862958  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.140.136:65535 -> 152.2.15.62:3456
12/21-09:06:03.874131  [**] High port 65535 tcp - possible Red Worm -
traffic [**] 152.2.15.62:3456 -> MY.NET.140.136:65535
12/21-09:06:03.894669  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.140.136:65535 -> 152.2.15.62:3456
12/21-09:06:04.105020  [**] High port 65535 tcp - possible Red Worm -
traffic [**] 152.2.15.62:3456 -> MY.NET.140.136:65535
12/21-09:06:04.165677  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.140.136:65535 -> 152.2.15.62:3456
12/21-09:06:09.109062  [**] High port 65535 tcp - possible Red Worm -
traffic [**] 152.2.15.62:3456 -> MY.NET.140.136:65535
12/21-09:06:09.118452  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.140.136:65535 -> 152.2.15.62:3456
12/21-09:06:09.118704  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.140.136:65535 -> 152.2.15.62:3456
12/21-09:06:09.129527  [**] High port 65535 tcp - possible Red Worm -
traffic [**] 152.2.15.62:3456 -> MY.NET.140.136:65535
```

The following link graph shows the connections for port 3456 activity between the two internal hosts MY.NET.140.136(A) and MY.NET.140.47(B) to and from other hosts:



Dshield (www.dshield.org) shows a large number of scans for this port just days prior to the dates chosen for this analysis (Dec. 17-21, 2002). Following is the dshield count for this time period on port 3456:

```
      date count
12/13/2002 3
12/14/2002 13905
12/15/2002 1742
12/16/2002 1734
12/17/2002 967
12/18/2002 55
12/19/2002 11
12/20/2002 20
12/21/2002 15
```

Outgoing Nimda

There was one entry for a Nimda alert. This is probably a false positive because a host infected with Nimda would be scanning many different remote hosts. The rule is triggered by presence of content "cmd.exe".

High Port 65535 activity (7886 alerts)

Port 65535 can be a normally used client port, but it is also associated with several trojans. Most of the alerts in this category seem to be related to two activities: file sharing programs (winmx and the like) and traceroutes.

The following alerts show a traceroute that came from port 65535 (partial list):

```
12/17-01:00:51.210306  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33435
12/17-01:00:51.211734  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33436
12/17-01:00:51.213746  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33440
```

```
12/17-01:00:51.213878  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33441
12/17-01:00:51.217002  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33446
12/17-01:00:51.219878  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33448
12/17-01:00:51.222131  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33449
12/17-01:00:51.224488  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33450
12/17-01:00:51.227831  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33452
12/17-01:00:51.230134  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33453
12/17-01:00:51.236260  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33454
12/17-01:00:51.242225  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33455
12/17-01:00:51.248472  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33456
12/17-01:00:51.258554  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33457
12/17-01:00:51.261303  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33458
12/17-01:00:51.290273  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33460
12/17-01:00:51.310164  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33461
12/17-01:00:51.331823  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33462
12/17-01:00:51.349365  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.140.9:65535 -> 199.109.32.48:33463
```

Note that they are udp packets with a increasing destination port number. This is typical of traceroute and other mapping tools like Sam Spade.

Following is an example of winmx traffic (or response to a winmx probe):

```
12/17-00:43:29.200609  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.84.178:6257 -> 219.7.228.102:65535
12/17-00:43:29.448786  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.84.178:6257 -> 219.7.228.102:65535
12/17-00:43:29.698703  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.84.178:6257 -> 219.7.228.102:65535
12/17-00:43:29.711424  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.84.178:6257 -> 219.7.228.102:65535
12/17-00:43:29.976315  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.84.178:6257 -> 219.7.228.102:65535
```

Other notes of interest about the network:
MY.NET.162.167 appears to be a web and ftp server, due to the number of alerts relating to ports 80 and 20.

Incomplete Packet Fragments
MY.NET.91.249 either has a bad nework card, or is involved in nefarious activity. 6215 Incomplete Packet Fragment alerts came from this host. 155 portscan alerts also came from this host. More than 6000 of these alerts were to one host: 65.107.130.149 (sgi.mikeace.com) during the span of about one hour:
(partial listing):

```
12/19-17:45:42.397070 MY.NET.91.249:0 -> 65.107.130.149:0
12/19-17:45:42.440295 MY.NET.91.249:0 -> 65.107.130.149:0
12/19-17:45:42.447634 MY.NET.91.249:0 -> 65.107.130.149:0
12/19-17:45:42.469281 MY.NET.91.249:0 -> 65.107.130.149:0
12/19-17:45:42.477263 MY.NET.91.249:0 -> 65.107.130.149:0
```

While it is possible that a bad network card can cause fragmented packets to be sent onto the network, the short time frame involved, combined with the scans that originated from this host leads me to believe otherwise. Hostile packets can be fragmented in hopes of avoiding firewalls or IDS some of which do not take the time to reassemble fragmented packets before checking for hostile content. Without full packet logs it is impossible to know what data these packets contained, but this is certainly not normal IP behavoir and the host should be investigated and/or removed from the network.

IRC evil (976 alerts)
IRC traffic can be normal, but it can also be used by trojans and bots to communicate. Some hosts had other alerts (such as portscans and exploits) mingled with the IRC alerts, which makes them look suspicious.

Exploit code
There were 840 alerts regarding exploit code. Just because exploit code was received does not mean the host has been compromised. The exploit code rules check for very specific hex strings in the payload. Most of these exploits are buffer overflows, where the attacker tries to get the victim host to execute arbitrary commands with elevated privileges. 97 different hosts on the internal subnet were targets of exploit code. The top 3 received over 100 alerts for each. In several instances, the exploit code was received just after or during IRC activity. This would indicate that participating in IRC increases your risk of being attacked.

Queso fingerprint (1403 alerts)
Queso is an os fingerprinting tool. It sends data to a host, and based on the responses received it can guess the operating system that is running.

Statistics

```
Top ten source addresses from alerts:
  count IP address
   8949 MY.NET.84.169
   6241 MY.NET.162.67
   6214 MY.NET.91.249
   4963 MY.NET.122.118
   4542 150.163.200.98
```

```
   4510 MY.NET.111.231
   4509 MY.NET.111.235
   4495 MY.NET.111.232
   4477 MY.NET.111.230
   4442 MY.NET.111.219

Top ten destination addresses from alerts:
  count IP address
  22432 192.168.0.253
   8949 63.170.89.12
   6241 150.163.200.98
   6182 65.107.130.149
   4552 MY.NET.162.67
   4029 209.10.239.135
   3033 MY.NET.17.48
   2462 64.95.120.131
   2140 MY.NET.6.40
   1903 207.200.86.97

Top ten scanners (source address - total 882 unique hosts):
  count IP address
 894894 MY.NET.70.176
 378432 MY.NET.83.153
 337838 MY.NET.150.213
 259058 MY.NET.88.165
 249912 MY.NET.91.252
 207645 MY.NET.84.244
 180989 MY.NET.87.101
 171671 MY.NET.114.45
 160125 MY.NET.114.25
 159017 MY.NET.84.178

Top ten hosts being scanned (destination address - total 649280 unique
hosts):
  count IP address
   4982 204.183.84.240
   4030 217.226.96.16
   2768 65.107.130.149
   2714 68.59.5.162
   2001 24.82.159.127
   1915 129.72.130.110
   1814 62.42.90.230
   1630 68.6.140.135
   1518 66.169.61.222
   1474 68.13.89.63
```

Note: The fact that there are so many more unique *destination* addresses as compared to source addresses tells me that a lot of scanning is taking place from the protected network. At least two such examples have been shown above. If most of the scans were being received on the protected network, you would see many different source addresses and fewer destination addresses.

```
Scans by type:
  count protocol
```

```
3846970 UDP
 493724 SYN
    476 VECNA
    107 INVALIDACK
     96 NULL
     77 NOACK
     66 UNKNOWN
     32 FIN
      8 FULLXMAS
      7 XMAS
      5 SPAU
      4 SYNFIN
      2 NMAPID

Top ten hosts sending out-of-spec packets:
  count IP address
    416 148.63.177.245
    408 MY.NET.70.183
    373 194.106.96.8
    372 MY.NET.53.10
    329 MY.NET.53.84
    109 65.214.36.150
     93 203.59.28.96
     71 209.47.251.14
     67 209.47.251.30
     66 81.95.99.139
```

Host 65.107.130.149, sgi.mikeace.com (many fragments sent to this host--who are we attacking?)

```
Trying 65.107.130.149 at ARIN
Trying 65.107.130 at ARIN

OrgName:    XO Communications
OrgID:      XOXO

NetRange:   65.104.0.0 - 65.107.255.255
CIDR:       65.104.0.0/14
NetName:    XOXO-BLK-15
NetHandle:  NET-65-104-0-0-1
Parent:     NET-65-0-0-0-0
NetType:    Direct Allocation
NameServer: NAMESERVER1.CONCENTRIC.NET
NameServer: NAMESERVER2.CONCENTRIC.NET
NameServer: NAMESERVER3.CONCENTRIC.NET
NameServer: NAMESERVER.CONCENTRIC.NET
Comment:
RegDate:
Updated:    2002-02-04

TechHandle: DIA-ORG-ARIN
TechName:   DNS and IP ADMIN
TechPhone:  +1-408-817-2800
TechEmail:  hostmaster@concentric.net
```

```
OrgAbuseHandle: XCNV-ARIN
OrgAbuseName:   XO Communications, Network Violations
OrgAbusePhone:  +1-989-758-6860
OrgAbuseEmail:  abuse@xo.com

OrgTechHandle: XCIA-ARIN
OrgTechName:   XO Communications, IP Administrator
OrgTechPhone:  +1-703-547-2000
OrgTechEmail:  ipadmin@eng.xo.com
```

Host 64.230.128.142, HSE-Windsor-ppp250535.sympatico.ca (trojan activity)

```
whois -h whois.arin.net !net-64-230-128-0-1 ...

CustName:   Nexxia HSE
Address:    1149 Goyeau Street Windsor Ontario N9A 1H9
Country:    CA
RegDate:    2000-11-23
Updated:    2000-11-23

NetRange:   64.230.128.0 - 64.230.135.255
CIDR:       64.230.128.0/21
NetName:    HSEN207-CA
NetHandle:  NET-64-230-128-0-1
Parent:     NET-64-228-0-0-1
NetType:    Reassigned
Comment:
RegDate:    2000-11-23
Updated:    2000-11-23
```

Host 148.63.177.245, vsat-148-63-177-245.c189.t7.mrt.starband.net (top out-of-spec sender)

```
Trying 148.63.177.245 at ARIN
Trying 148.63.177 at ARIN

OrgName:    Spacenet, Inc.
OrgID:      SPAN

NetRange:   148.62.0.0 - 148.78.255.255
CIDR:       148.62.0.0/15, 148.64.0.0/13, 148.72.0.0/14, 148.76.0.0/15,
148.78.0.0/16
NetName:    SPACENET-SPAN
NetHandle:  NET-148-62-0-0-1
Parent:     NET-148-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1-MCL.STARBAND.COM
NameServer: NS2-MCL.STARBAND.COM
NameServer: NS1-MAR.STARBAND.COM
NameServer: NS2-MAR.STARBAND.COM
Comment:    For abuse from IP range: 148.63.0.0-148.63.255.255
148.64.0.0-148-64.255.255 148.78.0.0-148.78.255.255 Please send email
to abuse@starband.com
```

```
RegDate:    2000-05-31
Updated:    2002-12-06

TechHandle: FM173-ARIN
TechName:   Miller, Fred
TechPhone:  +1-703-848-1108
TechEmail:  fred.miller@spacenet.com
```

Host 204.183.84.240 -no reverse dns available- (most scans received by this host)

```
Trying 204.183.84.240 at ARIN
Trying 204.183.84 at ARIN
Sprint SPRINT-BLKC (NET-204-180-0-0-1)
                                  204.180.0.0 - 204.183.255.255
Consult Dynamics, Inc. SPRINT-CCB75F2 (NET-204-183-80-0-1)
                                  204.183.80.0 - 204.183.95.255
Ashby & Geddes ASGE001-204-183-84 (NET-204-183-84-0-1)
                                  204.183.84.0 - 204.183.84.255
whois -h whois.arin.net !net-204-183-84-0-1 ...

OrgName:    Ashby & Geddes
OrgID:      ASHBYG

NetRange:   204.183.84.0 - 204.183.84.255
CIDR:       204.183.84.0/24
NetName:    ASGE001-204-183-84
NetHandle:  NET-204-183-84-0-1
Parent:     NET-204-183-80-0-1
NetType:    Reassigned
Comment:
RegDate:    1998-09-30
Updated:    1998-09-30

TechHandle: AG89-ARIN
TechName:   Geddes, Ashby
TechPhone:  +1-302-654-1888
TechEmail:  dns@dca.net
```

Host 150.163.200.98, moema.cptec.inpe.br (most alerts received from this external host)

```
Trying 150.163.200.98 at ARIN
Trying 150.163.200 at ARIN

OrgName:    Instituto Nacional de Pesquisas Espaciais
OrgID:      INDPE

NetRange:   150.163.0.0 - 150.163.255.255
CIDR:       150.163.0.0/16
NetName:    INPE-ANSP
NetHandle:  NET-150-163-0-0-1
Parent:     NET-150-0-0-0-0
NetType:    Direct Assignment
```

```
NameServer: GIOTTO.DPI.INPE.BR
NameServer: DIXIT.ANSP.BR
NameServer: YABAE.CPTEC.INPE.BR
NameServer: PATROA.DEM.INPE.BR
Comment:
RegDate:    1991-05-20
Updated:    2002-03-19

TechHandle: AM1110-ARIN
TechName:   Montes, Antonio
TechPhone:  +55-12-3945-6538
TechEmail:  montes@lac.inpe.br
```

DNS and whois info obtain with Sam Spade (www.samspade.org) and Unix tools dig and whois.

Analysis process

In a word: grep, grep and more grep. I concatenated the files of each type together and processed each with the standard unix tools grep, awk, sed, cut, uniq. The analysis was done on a Linux box running Suse 8.0.

I tried to use SnortSnarf, but it kept dying at some point while churning through the large alert file. I have used SnortSnarf successfully in the past, but apparently there was some corruption in the large file that kept it from processing.

The top ten lists were created with commands such as:

```
awk '{print $4}' scans | cut -d : -f 1 | sort | uniq -c | sort -rn >
scans.srcip.count
```

When data on a specific host or port was needed, grep was used to extract only the relevant lines:

```
grep 'MY\.NET\.105\.48' alerts | more
```

To answer such questions as, "which hosts caused IRC alerts?":

```
grep IRC outgoing | cut -d \] -f 3 | cut -d : -f 1 | sed s/\ //g | sort
| uniq -c |sort -rn
```

The following Perl script was written to help determine which hosts were talking to the same external hosts. It creates a file for each internal host, then puts all external hosts contacted in the file. This creates many files in one directory which can then easily be grepped for an ip address to show which internal hosts have been talking to it.

```
#!/usr/bin/perl -w

while (<>) {
  if (/.+\s\->\s/) {
    @line = split(/\s\->\s/, $_);
    $src = $line[0];
    $dst = $line[1];
```

```
    @s2 = split(/:/, $src);
    @d2 = split(/:/, $dst);

    $data{$s2[0]}{$d2[0]}++;
  }
}

foreach $k (keys %data) {
  open(FILE, ">$k");
  foreach $v ( keys %{ $data{$k} } ) {
    print FILE "$v\n";
  }
  close(FILE);
}
```

The basic syntax of the scripts used to process the files came from examples by Chris Baker. (http://www.sans.org/y2k/practical/Chris_Baker_GCIA.zip)

Last Updated: May 26th, 2017

Upcoming Training

Security Operations Center Summit & Training	**Washington, DC**	**Jun 05, 2017 - Jun 12, 2017**	**Live Event**
SANS Houston 2017	**Houston, TX**	**Jun 05, 2017 - Jun 10, 2017**	**Live Event**
SANS Columbia, MD 2017	**Columbia, MD**	**Jun 26, 2017 - Jul 01, 2017**	**Live Event**
SANS London July 2017	**London, United Kingdom**	**Jul 03, 2017 - Jul 08, 2017**	**Live Event**
SANSFIRE 2017	**Washington, DC**	**Jul 22, 2017 - Jul 29, 2017**	**Live Event**
SANSFIRE 2017 - SEC503: Intrusion Detection In-Depth	**Washington, DC**	**Jul 24, 2017 - Jul 29, 2017**	**vLive**
SANS San Antonio 2017	**San Antonio, TX**	**Aug 06, 2017 - Aug 11, 2017**	**Live Event**
SANS Boston 2017	**Boston, MA**	**Aug 07, 2017 - Aug 12, 2017**	**Live Event**
SANS Adelaide 2017	**Adelaide, Australia**	**Aug 21, 2017 - Aug 26, 2017**	**Live Event**
SANS Virginia Beach 2017	**Virginia Beach, VA**	**Aug 21, 2017 - Sep 01, 2017**	**Live Event**
SANS Network Security 2017	**Las Vegas, NV**	**Sep 10, 2017 - Sep 17, 2017**	**Live Event**
SANS vLive - SEC503: Intrusion Detection In-Depth	**SEC503 - 201709,**	**Sep 11, 2017 - Oct 18, 2017**	**vLive**
Baltimore September 2017 - SEC503: Intrusion Detection In-Depth	**Baltimore, MD**	**Sep 25, 2017 - Sep 30, 2017**	**vLive**
SANS Baltimore Fall 2017	**Baltimore, MD**	**Sep 25, 2017 - Sep 30, 2017**	**Live Event**
SANS London September 2017	**London, United Kingdom**	**Sep 25, 2017 - Sep 30, 2017**	**Live Event**
Community SANS Scottsdale SEC503	**Scottsdale, AZ**	**Oct 02, 2017 - Oct 07, 2017**	**Community SANS**
SANS October Singapore 2017	**Singapore, Singapore**	**Oct 09, 2017 - Oct 28, 2017**	**Live Event**
Community SANS Boston SEC503	**Boston, MA**	**Oct 09, 2017 - Oct 14, 2017**	**Community SANS**
Community SANS Ottawa SEC503	**Ottawa, ON**	**Oct 16, 2017 - Oct 21, 2017**	**Community SANS**
SANS Seattle 2017	**Seattle, WA**	**Oct 30, 2017 - Nov 04, 2017**	**Live Event**
SANS San Diego 2017	**San Diego, CA**	**Oct 30, 2017 - Nov 04, 2017**	**Live Event**
San Diego Fall 2017 - SEC503: Intrusion Detection In-Depth	**San Diego, CA**	**Oct 30, 2017 - Nov 04, 2017**	**vLive**
SIEM & Tactical Analytics Summit & Training	**Scottsdale, AZ**	**Nov 28, 2017 - Dec 05, 2017**	**Live Event**
SANS Cyber Defense Initiative 2017	**Washington, DC**	**Dec 12, 2017 - Dec 19, 2017**	**Live Event**
SANS OnDemand	**Online**	**Anytime**	**Self Paced**
SANS SelfStudy	**Books & MP3s Only**	**Anytime**	**Self Paced**