GIAC Certified Intrusion Analyst (GCIA) Practical Assignment Version 4.1

Yuan Fan, CISSP
hifanfan@yahoo.com
yfan@arcsight.com
Cupertino, CA 95014

Submission Date: 04/12/2005

SANS Golden Gate
San Francisco, CA 11/2004

Table of Contents

Part One: Executive Summary

Introduction:

As part of the effort to improve the current whole ABC universities network security. A security audit has been done based on 3 days of Snort IDS's log files, which contains 551,968 Snort alerts and 7,605,767 scan alerts as well as 29,623 out of spec logs. Analysis reports have been done and below are some conclusions and recommendations.

Conclusion:

1. There is strong evidence that some of the machines in the internal network have been compromised and are actively scanning other machines in the network.
2. There is evidence that an immediate review and change of current firewall rules settings are necessary. IDS rules need to update as well.
3. The current security situation in the campus is not really in a good shape, more investigation/actions need to be done and security device/tools may need to gain to improve security.

Recommendation

1. Take down some of the compromised machines immediately to stop further attacks or worm propagation by these machines. Also more investigation needs to be done on these machines.
2. Review and change/add firewall rules to prevent/improve current situations. Updating the Snort rules is also necessary.
3. Get more logs including TCPDUMP raw logs to narrow down the analysis and do more correlation to gain more confidence.
4. If possible, deploy more IDS including host based IDS, and deploy a security alert correlation/visualization/report management tool (whether commercial or open source) to improve/help the current situation.
5. Certain policy changes need to be done for internal network security including anti virus, important asset protection, OS patch enforcement, file sharing etc.

Part Two – Detailed Analysis

1. Data Used

I used the following data from April 20 to April 22, 2004 to do security auditing and analysis.

scans.040420.gz scans.040421.gz scans.040422.gz
alert.040420.gz alert.040421.gz alert.040422.gz
oos_report_040417.gz oos_report_040418.gz oos_report_040419.gz

Since there appeared some date mismatch between the oos filename as well as the real record inside; I picked the file which contains the event data in the same time range from April 20 to April 22.

For ease of analysis, I concatenated the alert logs to alert3days.txt, similar to oosAll.txt.

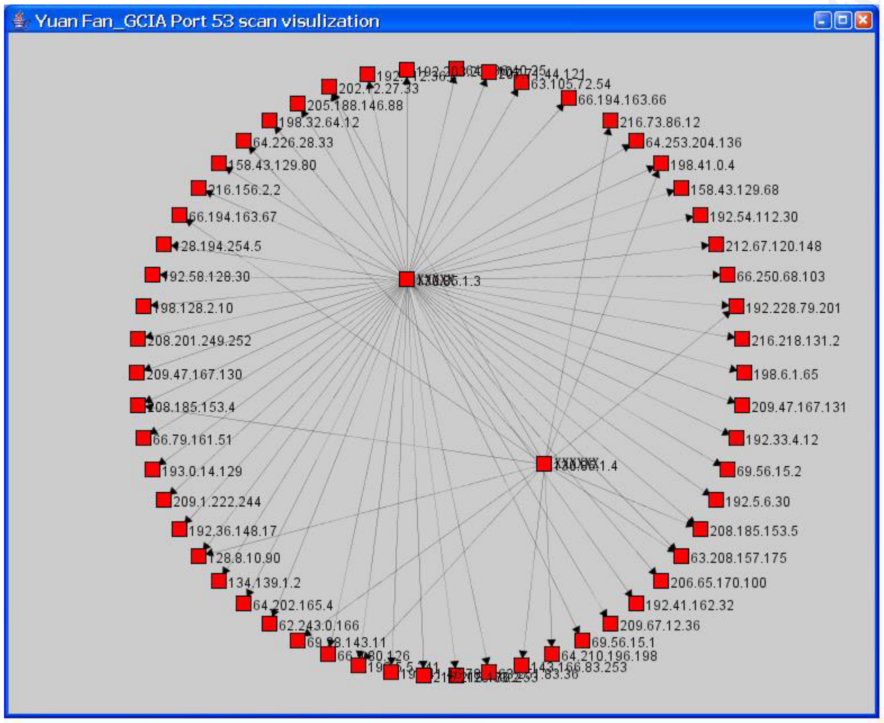
2. Network Spots

Based on the alert logs and OOS logs, we found the following spots/roles inside the network. It could be more accurate if we had raw dump file which includes the payload. But some real payload info inside the OOS logs did give us some confidence on this, for instance: MY.NET.70.254:80 in OOS logs had some Http headers inside.

MY.NET.1.3	DNS Server
MY.NET.1.4	DNS Server
MY.NET.1.5	DNS Server
MY.NET.12.6	SMTP Server
MY.NET.6.7	POP3 Server
MY.NET.5.67	Web Server
MY.NET.24.44	Web Server
MY.NET.70.254	Web Server
MY.NET.24.27	FTP Server
MY.NET.24.47	FTP Server
MY.NET.30.3	Netware Server
MY.NET.30.4	Novell Netware Remote Manager

Correlation:
Some of these results also can be correlated to the findings by Brett Hutley (GCIA) in his GIAC practical assignment paper. [Refer: 5]

Scan data can sometimes give us more insight during an investigation than alert logs. Even though network security attacks have been evolving for more than 10 years, the typical steps of scanning, finding target, exploiting vulnerabilities and taking control still apply in lots of cases. Further more, we found that scan data give important exposure in worm propagation case. For example, I used simple_ip_port_count.pl found that port 53 was the top destination port in scan data, then I made a perl script to convert the scan data to pajek .net data format [Refer: 31] and then wrote some java code which make use of the JUNG – Java universal network/graph framework [Refer: 6] to visualize the scan as below:



After visualization of just part of the scan data in the first day, we can clearly see that it is unusual for these two servers to give so many random IP 53 queries. In following chapter “detection one” we will describe details about this.

3. Detection one – Automated Massive UDP scan on port 53

Description of attack:

There is a massive UDP attempt on port 53 in the Snort log scans 040420, and all this traffic originated from an internal DNS server. Even though getting the real payload of the packet would help us to dig into it further, based on the current data we had, it has to link us to the bind name server vulnerabilities attempt at port 53. A snippet below explains:

```
> grep MY.NET.1.3:61408 scans.040420 |head -n 30
Apr 20 13:00:32 MY.NET.1.3:61408 -> 128.63.2.53:53 UDP
Apr 20 13:00:32 MY.NET.1.3:61408 -> 128.8.10.90:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 66.9.80.126:53 UDP
Apr 20 13:00:32 MY.NET.1.3:61408 -> 198.41.0.4:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 192.5.5.241:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 192.112.36.4:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 128.194.254.5:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 63.208.157.175:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 209.1.222.244:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 192.54.112.30:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 64.236.40.25:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 195.41.46.78:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 209.67.12.36:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 192.58.128.30:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 66.79.161.51:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 63.105.72.54:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 209.47.167.130:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 208.201.249.252:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 63.251.83.36:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 64.253.204.136:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 205.188.146.88:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 62.243.0.166:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 216.218.130.2:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 216.156.2.2:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 216.218.131.2:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 192.41.162.32:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 134.139.1.2:53 UDP
Apr 20 13:00:31 MY.NET.1.3:61408 -> 193.0.14.129:53 UDP
Apr 20 13:00:32 MY.NET.1.3:61408 -> 69.56.15.1:53 UDP
Apr 20 13:00:32 MY.NET.1.3:61408 -> 208.185.153.5:53 UDP
…
```

Reason this detect was selected:

Normally just a horizon port scan should not cause so big an alarm, but this time a big 53 port scan from a DNS server really worries me. At first glance it also reminds me of the 53_worms like ADM_worm or Li0n worm [Refer: 7,8,9], but since it is UDP traffic, it is at least not directly related. Of course we need to investigate more, especially on the real payload (since with the current log we don't have payload info), but the current observation is still worth investigating. Not to mention that in the Snort logs 53 is the top port targeted and MY.NET.1.3 and MY.NET.1.4 are two of the top malicious sources that we caught.

Detect was generated by

We gained information from the scan data originating from the two DNS servers mentioned above. It is interesting that we didn’t see any specific Alert log on this. There were some “Possible Trojan server activity” messages in alert logs and it seems that it tried to catch the port 27374 activity. This appeared to be a custom rule, and Port 27374 is the port which Li0n v3 and Ramen worms used to let others download a copy of the worm code. On the other hand, we can see that many “Possible Trojan server activity” alerts are really misleading, but some of them are really suspicious.

Probability that the Source Address Was Spoofed

The source address was probably not spoofed in this case. Based on the observation of the source port occurrence as well as the destination IP addresses space, this is more like an automated script who did this behavior. There could be a concern of a DOS attack in which someone generates a packet with source IP MY.NET.1.3 so that all the reset/response packets will be sent to MY.NET.1.3, but if I were trying to DOS attack MY.NET.1.3 DNS in this case, I would use source port 53 instead of port 61408; that way, all the traffic response back will hit 53 port of the DNS server.

Attack Mechanism

Based on the log we saw, it is a massive 53 port scan and most likely it is a DNS bind named version attempt. The attack was most likely generated by an automated script and targeting UDP port 53 on random IP address range. CVE 2001-0010 and CVE 1999-0009 describe the buffer overflow attack against certain Bind versions. [Refer: 11, 12] After gaining the bind version of DNS, the next step could be identifying what vulnerability is associated and using the right exploit against the victim. Most versions of bind will, by default, respond to the query while bind version 9 is an exception. [Refer: 35].
There is also an interesting pattern worth mentioning:

```
>grep ‘82.196.5.2:’ scans.040420
Apr 20 22:19:33 MY.NET.1.3:61586 -> 82.196.5.2:53 UDP
Apr 20 22:30:38 MY.NET.1.4:32788 -> 82.196.5.2:53 UDP
Apr 20 23:08:57 MY.NET.1.3:61586 -> 82.196.5.2:53 UDP
Apr 20 23:08:57 MY.NET.1.3:40391 -> 82.196.5.2:53 SYN ******S*
```

We could gain more confidence and information if we can investigate on the UDP packet to see the payload it contains targeting the remote host on port 53.

Even though it is not directly related, this novel massive 53 attempt reminds us of the need for further investigation of worms which are related to bind name server vulnerabilities. There are variants of different worms against bind name server vulnerabilities; there are lots of common mechanisms with this attack. Furthermore, they even share some code between them. This attack targets remote bind name server vulnerabilities at port 53 on Linux specifically. After gaining access, it will create a backdoor account, send critical info to a preset email for the "worm owner" and get exploit code and start new spreading again. For example, both ADM worm and Li0n worm are very similar in nature, they both target Linux systems with bind exploit vulnerabilities. Li0n uses a couple of scripts as well as binary code to exploit bind server vulnerabilities and then after it gains access, it will email itself the /etc/passwd and /etc/shadow (depending on the version of the Li0n worm this behavior acts a little bit different), but a main action after email will use lynx to download a copy of the worm code from a website to spread later on. [Reference: 7, 8, 9]

Correlations

John's post "LOGS: GIAC GCIA Version 3.3 Practical Detect" noticed some 53 activity and list of the Trojan on these ports. [Refer: 32]
http://www.dshield.org/pipermail/intrusions/2003-May/007803.php

Al Williams discuss this "Possible trojan server activity" and 27374 activity in his GCIA paper in page 56 [Refer: 33]

Ian mentioned a similar but not exact same behavior of DNS named version attempt Snort alert in following link [Refer: 34]:
http://www.dshield.org/pipermail/intrusions/2004-July/008200.html

Reference:
SANS has a good advisory about Li0n worm in http://www.sans.org/y2k/lion.htm, and internet storm center is a good place to reference:
http://isc.sans.org/port_details.php?port=53
WhiteHats Network Security Resource has a good analysis about Li0n and ADM worm [Refer 7, 8].

Evidence of Active Targeting

This does not appear to be an active targeting, but it is more a scanning/propagation. The IP address it was targeting was pretty random. On the other hand, MY.NET.1.3 and MY.NET.1.4 had been affected already before the date we took the current Snort logs, so certain Actions need to be taken immediately.

Severity

Severity = (criticality + lethality) - (system countermeasures + network countermeasures)

Criticality: 4
This is a critical threat, at least two DNS servers have most likely been compromised and they are actively propagating to internet/intranet.

Lethality: 3
We can not really decide in this case because of lacking of further more information. If it was a recon, once the vulnerability gets identified, buffer overflow attack could follow next.

System countermeasures: 1
We don't have a big confidence of this part right now, even though based on the current info it is not enough, but the vulnerability was found and released much earlier than 2004, the system OS/IDS seem to be lacking the patch/protection process.

Network countermeasures: 1
We see little strength about network countermeasures, seems the firewall need to be reviewed to block certain activity. For example: internet user can directly have SMB query of intranet machines.

Over all score = (4+3) – (1+1) =5

Defensive Recommendation

DNS server usually is hotspot in the internet. We should investigate the hosts MY.NET.1.3 and MY.NET.1.4 and apply the OS patches especially for bind vulnerability immediately if it is a vulnerable version. Also apply the rules for DNS related threats immediately for Snort if this is not done yet. These two originators (1.3 and 1.4) need to be taken down if possible and more investigation need to be done, including /etc/password, /etc/inetd.conf. We need to close unnecessary ports which are not needed for DNS server needs and get rid of the service package from OS.

Detection Two: Tiny Fragments - Possible Hostile Activity

Description of attack:

In TCP/IP protocol, the IP layer sometimes needs to dissect the packet to make sure it adopt to the different network, and the end destination needs to reassemble the small pieces into a big packet back. IDS usually listen in the middle of this transmission and investigate the little packet. Since it is very resource consuming to keep the state of the transmission and reassembly of the packet, it is possible for the attacker to use tiny packets which overlap each other to avoid IDS detection.

Reason this detect was selected

IDS fragmentation evasion is a very common problem in network security world. Also this alert “Tiny Fragments - Possible Hostile Activity” was one of the biggest traffic inside these 3 days alert logs, it is worth to look into this suspicious activity.

Detect was generated by

This detect was generated by Snort rule:

```
alert tcp any any -> any any (minfrag: 256; msg: "Tiny fragments detected, possible hostile activity";)
```

Whenever the TCP packet size is less than 256, this rule will get triggered. Note minfrag is configurable. A usual device wont make the fragment less than 256 bytes while attackers could use much small fragmentation than 256 for the IDS evasion.

Address Spoofing Probability

In our case, the source IP was not likely to be spoofed, we also correlated “null scan” alert as well as a big scan by the same IP in the scan log as well. If it really wants to get the ACK back within TCP 3-way handshake, then it has to use the real IP instead of the fake one.

```
grep "209.164.32.205:" alert3days.txt
04/22-18:59:12.214492  [**] Null scan! [**] 209.164.32.205:0 -> MY.NET.97.55:0
04/22-18:59:12.499631  [**] Null scan! [**] 209.164.32.205:0 -> MY.NET.97.55:0
04/22-19:39:47.357768  [**] spp_portscan: portscan status from 209.164.32.205: 1
connections across 1 hosts: TCP(1), UDP(0) STEALTH [**]
04/22-18:59:13.044118  [**] Null scan! [**] 209.164.32.205:0 -> MY.NET.97.55:0
04/22-19:39:53.675406  [**] spp_portscan: portscan status from 209.164.32.205: 1
connections across 1 hosts: TCP(1), UDP(0) STEALTH [**]
```

But generally it is not impossible for an attacker to use spoofed address sometimes. For example: if the attacker knows the IDS inside used stateful packet reassembly mechanism, he could send millions of tiny fragmented packet with spoofed address to make the IDS consume all its resources!

Attack Mechanism

Instead of sending an attack in one packet, the attacker chooses to use fragmentation to send many small packets with the same effect. To catch the attack, most of the IDS in the world (that is signature based IDS) needs to check the header and/or payload to look for certain patterns. That means they have to reassemble the small packet if they are in fragmentation. There are two problems that arise here:

1. Keeping these entire small fragmentation packets in memory and reassembling them is very resource/time consuming, which in many cases could suffer “denial of service”.
2. Even if it is strong enough and did the full packet reassembly and investigation, the IDS still could suffer from the fragmentation overlapping and fragmentation TTL time-out attack.

For example: If the IDS wants to catch /default.ida, and if the attacker sends a packet with that, it will be caught. Instead, if it sends /,d,e,f,a,u,l,t,.,i,d,a in each small packet, and if the IDS didn’t reassemble the fragmented packets, it can evade the detection.If the IDS is doing reassembly, then the attacker may send /deAfault.ida in each fragmentation, so the IDS will think this signature not match. But when the attacker sends A, he can set the TTL so that the packet containing A will reach IDS but expire before reaching the victim, so the final packet will reach the victim being reassembled as “/default.ida”. [Refer: 15]

Correlations

In 1998 Thomas Ptacek and Timothy Newsham wrote an IDS paper to describe the IDS evasion techniques including fragmentation in their IP Fragmentation chapter. http://secinf.net/info/ids/idspaper/idspaper.html

Dug Song was another name need to mention here. He wrote the tool called fragroute (Reference: 20) which implemented most of the attacks mentioned above for IDS/firewall testing purpose.

Securityfocus had a good article mentions about “IDS Evasion Techniques and Tactics”. http://www.securityfocus.com/printable/infocus/1577

Maarten Van Horenbeeck’s GCIA paper below mentions the Fragmentation overflow attack which is similar and related to the topic.
http://www.daemon.be/~maarten/Maarten_Vanhorenbeeck_GCIA.pdf

Evidence of Active Targeting

```
E:\fan\GCIA\data>grep "Tiny Fragments - Possible Hostile Activity" alert3days.txt
|cut -d ']' -f3 |cut -d ' ' -f4 |sort |uniq -c
```

```
  …
    7 MY.NET.97.30
    7 MY.NET.97.58
   27 MY.NET.97.20
   41 MY.NET.12.6
  657 MY.NET.81.116
 1654 MY.NET.97.55
 1974 MY.NET.97.43
```

From distribution above, we can see it is not targeting one target for sure, but MY.NET.97.43 and MY.NET.97.55, MY.NET.81.116 large number of alerts, which is very suspicious. If there is a security tool designed to monitor this network, these two target IPs are recommended to be put into a heat list for correlation/analysis later on.

Severity

Severity = (criticality + lethality) (system countermeasures + network countermeasures)
Criticality: 3.5
Even though it is a evil scan, it is still with medium criticality so far.

Lethality: 3.5
If it gets IDS evaded and gets to the target, it will depend on what the real packet is for the real damage, we can not get accurate estimation here yet.

System countermeasures: 2
We don't really know the strength of system defense yet, but from the alerts generated by the two heated up IP addresses mentioned above, we are not in very comfortable situation.

Network countermeasures: 3
It seems that IDS was able to detect this since this was a simple rule. More correlation needs to be done to really detect/mitigate the attacks.

So total score:
Severity = (3.5+3.5)-(2+3)=2

Just judged by this alert it gives low severity, however, more investigation needs to be done on the packet and host mentioned above. For example: what are the real packets it tries to evade the IDS? What is the service it is trying to targeting?

Defensive Recommendation

Fragmentation IDS evasion was a very common problem before and will still the trouble for IDS for a while (from personal point of view); the reason has been main listed in attack mechanism above. From defense point of view, we need to make sure the IDS can handle stateful packet investigation but not DOS attack, also it is desirable to have some penetration tool like fragroute above to do a self pen-test and find the problems before the attacker finds it.

Detect Three: When Bot (Agobot like) starts exploit LSASS vulnerabilities and scan mydoom and other backdoors.

Description of detect

The alert "Possible sdbot floodnet detected attempting to IRC" appeared 57 times in alert3days.txt and targeting outside port 7000. Sdbot was a type of Trojan, after it gets control of the computer (through mail attachment method for example), will create an IRC channel connecting back to an IRC server to download executables or get commands for further more recon/exploits. We noticed that this time the machine who detected with this alert starting actively scan the backdoors and LSASS
Vulnerability (MS04-011) [Refer: 17]
CAN-2003-0533[Refer: 18] describes that this attack is based on buffer overflow remote exploit in Local Security Authority Subsystem Service (LSASS) on most popular windows platforms.

Reason this detect was selected

Sdbots almost always come with command-control ability which the attacker could control the compromised machine and do malicious scan/attacks against the network. From above we can clearly see that it starts a malicious scan against LSASS vulnerabilities as well as known backdoors like MyDoom(3127) and Bagle worm(2745).

Detect was generated by

This "sdbot" message appeared generated by a Snort custom rule. From http://coders.meta.net.nz/~perry/irc.rules
We can get some clue about how this one type of IRC rule works for Snort:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 6660:7000
(content: "USER ";content: " 0 0 "; nocase;msg: "Possible sdbot
floodnet detected attempting to IRC"; classtype:misc-activity;)
```

From above we can see that it tries to catch the packet by both the destination port and the content in case insensitive way. Even though the possibility of false alarm is there, but considering the malicious behavior of these bots as well as the population of the variants of bots in the world, this rule is worthwhile to have, from my personal point of view. If we can have a manner to correlate the behavior including this IRC as well as later on malicious scan in network, that would be even better.

Probability the source address was spoofed

It is not really possible in reality that the source was spoofed in this case, since the IRC chat has to be done after the 3-way handshake, and this rule is catching the content as well after TCP handshake is done. If there is really a machine that is sending fake packets including those packets without any TCP handshake, then that malicious machine is not going to get any packets back from IRC server since the reset packet will be hitting this spoofed address. There is really not a lot of sense to have the motivation to do this unless it wants to have a DOS attack, but by looking at the events rates and total events, it doesn't seem to be the case.

Attack Mechanism

By the time of the data gathered, there are already hundreds of variants of bots in the world, so what describes below is mostly giving a common/typical explanation.
A "bot" Trojan could arrive to the machine in many ways. One of the most common ways is by email attachment executables. Once the user clicks the executable it will hide the Trojan to windows system directory with a similar name to some existing Microsoft files. For example: %system%\iexplore.exe. It will also most likely modify the registry so that it can perform tasks later on. The location need to be checked include the following [Reference: 19]:
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce

This Trojan usually contains its own IRC client functionality and will connect back to the IRC server so that the attacker can remotely control the Trojan to do the following things (but not restrict to following):
1. Download/execute executable files
2. Perform malicious scan/exploit on certain known vulnerabilities
3. Update Trojan itself.
…

Correlations

The "Handler's Diary April 1st 2004" in SANS internet storm center first mentioned this malicious activity and predicted a new type of bot/worm activity scanning ports: 1025, 135, 139, 2745, 3127, 445, 6129, 80, 8080. Note the pattern of ports here is similar to what we discovered above except 8080. In this

diary it also provided the information which someone mentions this could be a modified W32/Agobot-EM that makes this.
[Reference: http://isc.sans.org/diary.php?date=2004-04-01]
Just in the next day, Bill McCarty mentioned that he noticed same pattern except tcp/8080 in the discussion: [Dshield] Mydoom probe in Handler's Diary for April 1, 2004. [Reference: http://lists.sans.org/pipermail/list/2004-April/047569.html]

Evidence of Active Targeting

This was not an active targeting on any specific host. But the scan above was actively targeting the backdoor port as well as RPC, LSASS vulnerabilities. It is interesting to see how densely on the ports the bot machine was targeting.
We could get more confidence if we can look into some payload these events generated. For example: what it tries to sending to 80 port? What is the packet payload for port 1025?

```
> grep 'sdbot' alert3days.txt |cut -d ']' -f4 |cut -d ':' -f1 |sort |uniq -c |sort
      1  MY.NET.43.10
      1  MY.NET.69.155
      1  MY.NET.69.210
      2  MY.NET.112.189
      2  MY.NET.80.119
      3  MY.NET.153.195
      8  MY.NET.112.193
     14  MY.NET.17.45
     25  MY.NET.84.186

> grep "MY.NET.17.45" scans.* |cut -d ' ' -f6 |cut -d ':' -f2 |sort |uniq -c |sort
…
  24074 5000
  24331 3410
 103896 80
 132569 139
 136111 6129
 140531 3127
 144928 445
 149320 1025
 159044 2745
 164232 135
```

Severity

Criticality: 5

The bot is randomly actively targeting web server, mydoom backdoor, LSASS exploits and so on, we can not put low criticality here just because it is random targeting.

Lethality: 5
The damage will be very severe if any one of the exploit gets successfully compromised or any backdoors found there.

System countermeasures: 1
There is little evidence that there is any host based IDS or anti virus software applied to this system.

Network countermeasure: 2
We had a rule that could almost detect a bot, but that was it. It would be much nicer if we could detect certain attacks against the web or LSASS vulnerabilities. As we see from the correlation, both the bot and the some exploits are pretty up-to-date. Expecting networks to have a good countermeasure at that time is not very realistic.

Severity = (criticality + lethality) (system countermeasures + network countermeasures) = (5+5)-(1+2) =7

Actions need to be taken on those machines immediately and detailed analysis or possibly full recovery of the OS needs to be done.

4. Network Statistics

To automate and simplify the network statistic process, a perl script simple_ip_port_count.pl in appendix has been made to analyze the Snort data files. This script was refactored based on the apr.pl by Frans J.H. Kollee (GCIA) [Reference: 23]. Originally the script was built for Snort alert, and the refactoring mainly involved changing so it can work generally for both scan and alert logs and some small bug fixes. Also it has certain parameters with different functions like Top n talkers, Top 10 Services etc. For example below we analyze top 10 within all the scan data from 040420 to 040422:

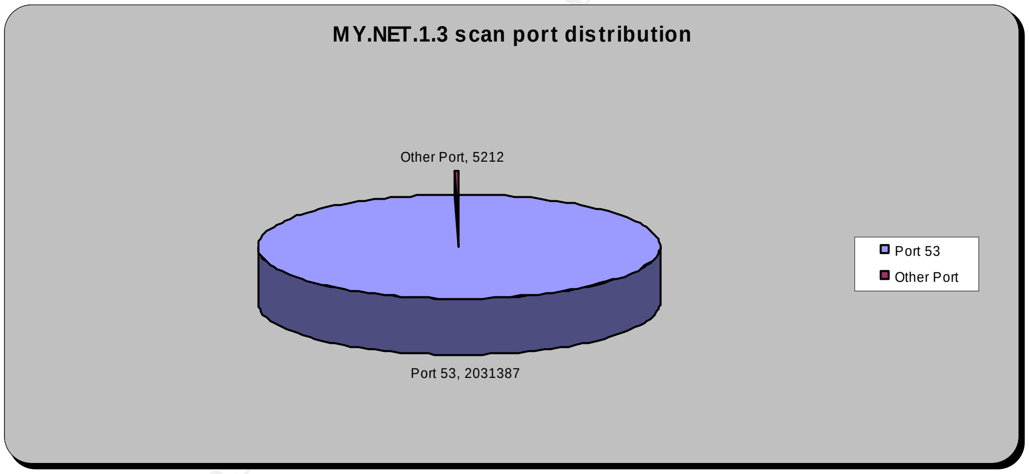
```
# perl simple_ip_port_count.pl -a -n -y allscans.txt
====== Reports sort by Alerts ======
======2036599===MY.NET.1.3=======
======1179161===MY.NET.17.45=======
======713579===MY.NET.112.189=======
======655139===MY.NET.81.39=======
======589543===MY.NET.1.4=======
======447521===MY.NET.84.186=======
======409747===MY.NET.112.193=======
======114202===MY.NET.69.210=======
======94577===MY.NET.69.214=======
======73028===MY.NET.97.12=======
------------- Top Destination Ports ---------
```

```
-----Port: 53 ---> 2616678 Times-----
-----Port: 135 ---> 1691392 Times-----
-----Port: 80 ---> 422275 Times-----
-----Port: 2745 ---> 332213 Times-----
-----Port: 6129 ---> 326309 Times-----
-----Port: 1025 ---> 294460 Times-----
-----Port: 445 ---> 284050 Times-----
-----Port: 3127 ---> 272470 Times-----
-----Port: 139 ---> 251053 Times-----
-----Port: 3410 ---> 132822 Times-----
```

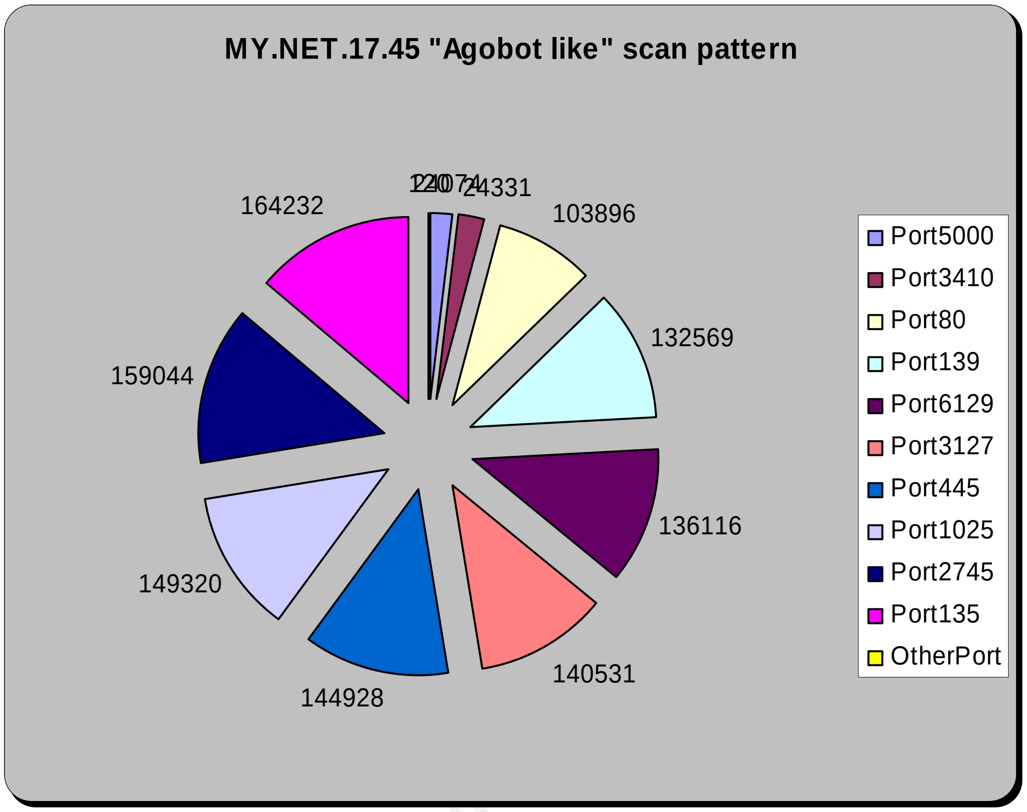
Top 5 talkers

From Scan data:

- MY.NET.1.3
  MY.NET.1.3 probed 84448 destination(s), 722 distinct port(s) (2036599 total scan alerts). Specifically, among those scan alerts 2031387 alerts are targeting on port 53 in the network, distribution pie chart below:



- MY.NET.17.45
  This “bot” machine probed 142819 destinations, 18 distinct ports (1179161 alerts total). It generated more than 99.9% of the scan on port 135,2745,1025,445,3127,6129,139,80,3410,5000, scan pattern are showing as below:



- MY.NET.112.189
  IP MY.NET.112.189 probed 530123 destinations, 1 distinct port: 135 (713579 alerts total). This is another one which had most focused port.
- MY.NET.81.39
  This host has similar pattern to MY.NET.112.189. MY.NET.81.39 probed 654445 destinations, 17 distinct ports (655139 alerts total). Mostly focus on port 135 as well.
- MY.NET.1.4
  This DNS server had very similar pattern to MY.NET.1.3 above.

From Alert Data

Top 5 Talkers are:

```
[yf@ylinux]# cat alert3days.txt |grep –v –i portscan |perl -pe 's/\[([^*]+)\]/\1/g' |cut -d ']' -f3 |cut -d ':' -f1 |sort -b |uniq -c |sort -rn |head -5
  21784  134.192.42.11
   3730  68.55.155.26
   3243  131.92.177.18
   3230  MY.NET.43.8
   3108  MY.NET.11.4
```

Top Destination Ports/Services:

From Scan data:
As we analyzed from the detections chapter above, it is not a surprise to see the following results:

- Port 53 (2616678 Times)
- Port 135 (1691392 Times)
- Port 80 (422275 Times)
- Port 2745 (332213 Times)
- Port 6129 (326309 Times)

From Alert data:

```
[yf@ylinux]# cat alert3days.txt |grep –v –i portscan |perl -pe 's/\[([^*]+)\]/\1/g' |cut -
d ']' -f3 |cut -d ':' -f3 |sort -b |uniq -c |sort -rn |head -5
```

- 51443 (25487 times)
  Usually this is a client port going out, but most people will run Novell netstorage server on port 51443.
- 80 (12215 times)
  WebDav vulnerability exploiting is also one of the activity that “agobot like” bot will do [Reference: 24].
- 65535 (9997 times)
  65535 was known as Adore worm/RC1 trojan backdoor [Reference: 25].
- 524 (9030 times)
  524 was a service port of NCP (Novell Netware Core Protocol).
- 137 (5981 times)
  Famous netbios name service port, no need to say more.

Three Most suspicious external addresses:

- 131.234.100.43
  This host appeared to be down now. This was the controller side of “Agobot like” and it give command to MY.NET.17.45 who scanned lots of backdoors and other vulnerabilities. It would be interesting to look into the traffic between IRC client and server when the Snort alert generated.

- 213.180.193.68
  This Russia IP scanned host MY.NET.97.159 and MY.NET.97.65 and generated 36433 scan alerts. What is the motivation behind this? The scanned ports seem to be pretty random, but the targeting port host was in focus. This host also generated the Snort alert “TFTP - External TCP connection to internal tftp server” and “High port 65535 tcp - possible Red Worm – traffic”.

whois 213.180.193.68

```
% This is the RIPE Whois query server #2.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/db/copyright.html

inetnum:      213.180.192.0 - 213.180.193.255
netname:      COMPTEK-NET1
descr:        CompTek International/Yandex LLC
descr:        3, Gubkina str., Moscow, 117809
country:      RU
admin-c:      YNDX1-RIPE
tech-c:       YNDX1-RIPE
status:       ASSIGNED PA
notify:       noc@yandex.net
mnt-by:       YANDEX-MNT
changed:      wawa@comptek.ru 20020607
changed:      gvs@yandex-team.ru 20040625
source:       RIPE

route:        213.180.192.0/20
descr:        Yandex enterprise network
origin:       AS13238
notify:       noc@yandex.net
mnt-by:       YANDEX-MNT
changed:      wawa@comptek.ru 20010123
changed:      gvs@yandex-team.ru 20040625
source:       RIPE

role:         Yandex LLC Network Operations
address:      Yandex LLC
address:      40A Vavilova st.
address:      117333, Moscow, Russia
phone:        +7 095 9743555
fax-no:       +7 095 9743565
e-mail:       noc@yandex.net
trouble:      ------------------------------------------------------
trouble:      Points of contact for Yandex LLC Network Operations
trouble:      ------------------------------------------------------
trouble:      Routing and peering issues:  noc@yandex.net
trouble:      SPAM issues:                 abuse@yandex.ru
trouble:      Network security issues:     abuse@yandex.ru
trouble:      Mail issues:                 postmaster@yandex.ru
trouble:      General information:         info@yandex.ru
trouble:      ------------------------------------------------------
admin-c:      VLI1-RIPE
```

```
admin-c:     GVS-RIPE
tech-c:      KBG2-RIPE
notify:      noc@yandex.net
nic-hdl:     YNDX1-RIPE
mnt-by:      YANDEX-MNT
changed:     gvs@yandex-team.ru 20040625
source:      RIPE
```

- 220.197.192.39
  This is an IP belongs to China Unicom. This IP generated 1997 “EXPLOIT x86 NOOP” Snort alerts. It also generated 15889 scan alerts in Snort 3 days scan logs. One of the most interesting pattern in scan data is that it had a similar pattern to the ‘Bots’ in detection 3 but not exactly same. It is obviously targeting 2745, 6129 and 80 as the main target port. Is this machine controlled by a ‘Bots’ already, or actually it was controlled by an attacker with an automated tool? More data need to be analyzed and also more correlation work needs to be done.

```
grep  ‘220.197.192.39’ scan3days.txt |cut -d ' ' -f6 |cut -d ':' -f2|sort |uniq -c |sort
      1 1981
      1 23079
      1 25838
      1 51
      3 1025
      7 53
    102 3127
    104 139
   4973 80
   5206 6129
   5490 2745
```

Out Of Spec (OOS) files

Almost every oos record triggered by the following:

```
12****S* Seq: 0x34CA09E5  Ack: 0x0  Win: 0x16D0  TcpLen: 40
```

The ECN (Explicit Congestion Notification) bits were getting set. However, so far I could not see real evil activity here.

5. More Correlations - GCIA, CERT, BugTraq, Etc.

The Snort data logs I picked were almost the newest files I can get in http://isc.sans.org/logs. So far the only GCIA I found who used this same date range to analyze is Jorge D. Ortiz-Fuentes, however, I just realized after I finished this paper, we had 3 totally different angle on the 3 detections. Wouter Clarie analyzed same month’s data in his GIAC Certified Intrusion Detection Analyst Practical Assignment V4.1 [Reference: 22], even though I read his paper a little late, but was very glad to see the ‘Bots’ scan pattern finding was pretty similar. And Brett Hutley analyzed the alert "IRC evil - running XDCC" in his V4.1 GIAC practical assignment paper for Snort log from 040407 to 040411 [Refer: 5].

Internet storm center (http://isc.sans.org) is always a great place to correlate and reference things, especially for variants of worms/virus/bots and latest security trends.
In latest <<Symantec Internet Threat Report Volume VII>> [Reference: 26], it also emphasized the seeing of increasing threat of "Bots" (from end of 2003 to end of 2004)

6. Defense recommendation

1. Take down/isolate compromised machines and mitigate the threat immediately.
   Quite a few hosts are known to be compromised already. We need to isolate them and fix it as soon as possible. For example: MY.NET.17.45, MY.NET.1.3, and MY.NET.1.4

2. Apply security policy in the network
   - Patch management
     One of the core problems with all the threats here is we didn't apply the security patch or upgrades the OS which we should do. For example: if the bind name servers were upgrade to newer version, it could avoid the buffer overflow attack.
   - Strong password
     Variants of Bots including "Agobot" will try to exploit weak network sharing password. It's very important to have a strong password policy and auditing every once a while.
   - File Sharing to internet
3. Tune the firewall and IDS rules to be more effective.
   The Snort rule seems to be pretty out-of-date, we need to update the rules as well as upgrade Snort to latest version. Certain firewall rule need to apply, for example: why do we allow outside IP 137 port to communicate with internal Host 137 port? Can we please deny the traffic which generated from outside network targeting known Trojan port inside the campus network? If the firewalls can not perform stateful investigation of the packets, it maybe comes the time to install a firewall with that functionality.
4. Consider to get more security tools to help
   There are certain tools that could help a lot in the current situation. For example: install anti virus software and deploy host based IDS in important asset. It would be nice and probably important to have a security event correlation/report tool (whether open source or commercial) to manager such a big data flow every day as well as do more intelligent/precise work.

5. Basic security knowledge/policy training and brainstorm for whoever uses the network.
   I kind of agree the view that the most vulnerability was not in network but from the person who are in and use the network. For example, even if you deploy a powerful anti-virus software, but people can disable it because they need to do something more freely. How will the security get better in this network?
6. Scheduled network vulnerability scan.
   Found the vulnerabilities before the hacker finds it!

Part Three Analysis Process

Two machines have been used to analyze Snort data files. Machine 1 is XP+Cygwin, Machine 2 is Debian linux with kernel 2.6.7. Perl, java and some unix tools are used for analysis and visualisation.
The perl script and java code should work consistently on both platform. The unix command used in this paper might have slight differences between the two platforms.

Potentially Compromised Internal Hosts

From above analysis we can see, MY.NET.1.3, MY.NET.1.4, MY.NET.17.45 had aroused enough interests and worth for us to do more investigation. There are certainly more machines having signs of being compromised based on the Snort data files we had.

Description of Analysis Process

In short, in first phase I used the simple_ip_port_count.pl and UNIX commands to get different Top N report from the Snort data files. Then I get some most suspicious aspect to drill down and try to correlate between different Snort logs.
For example:
Top destination port/service:

```
perl simple_ip_port_count.pl -a -n -y ..\data\scans.040420
====== Reports sort by Alerts ======
======554275===MY.NET.17.45=======
======511084===MY.NET.1.3=======
======187794===MY.NET.84.186=======
======175109===MY.NET.1.4=======
======171674===MY.NET.112.189=======
======130976===MY.NET.81.39=======
======72400===MY.NET.153.195=======
======27185===MY.NET.98.42=======
======21957===213.180.193.68=======
======19508===MY.NET.97.248=======
-------------- Top Destination Ports ---------
-----Port: 53 ---> 684404 Times-----
```

```
-----Port: 135 ---> 417071 Times-----
-----Port: 2745 ---> 111966 Times-----
-----Port: 80 ---> 107927 Times-----
-----Port: 1025 ---> 99008 Times-----
-----Port: 445 ---> 95646 Times-----
-----Port: 3127 ---> 93145 Times-----
-----Port: 6129 ---> 91891 Times-----
-----Port: 139 ---> 86750 Times-----
-----Port: 3410 ---> 48803 Times-----
```

After we visualize all the 53 activity with source/target, it is really novel. This report also put MY.NET.17.45 in a hot spot that remind me to dig more into the scan ports as well as the correlation into Snort alert logs, and finally we found the evil activity. While getting similar ideas to Ed Skoudis's "malware analysis template ". [Reference: 29] I took a brave to summarize my process/thoughts and put into a simple template for analysis (based on Snort's data files). It is very raw draft based on the log files like Snort generated. Just hope it can benefit a bit whoever interested.

Snort Scan/Alert log file analysis template:

Scan data analysis: Top Destination Port/Service get scanned	What services are related to the Destination port? Is it a known service port, or is it a known Trojan port? Or is it unknown and need more evidence yet… Is it a horizon scan or vertical? TCP or UDP?
Top target address/Port combination	Do we know Is this a server or a normal pc? Why these servers get so many interests in this port? Is there any other activity targeting to or from this machine? (correlate Alert log here)
Top scanner and behavior	What are the top scanners? Is it resides in local network, or from internet? Is it within TCP spec or out of spec.
Alert file: Top Alert names	What is the indication of this Alert name? What can we correlate from the scan data? Is it real alarm or more like a false alarm? Did the attacker address also have similar behavior to other destination host too? If they are what are they?
Top source IP/destination Port	Why so many Alerts go with the same

combinations	source IP/destination Port combination? What is the indication here? A worm or something else?
With one Alert lock down:	What are the destination port and data traffic? Is there any destination is known to be compromised and need to take further actions? What are the related events from Alerts and scan data?

All in all, analysis process may vary by different analyst, but how to quickly find the spot inside big amount the data is the key, after that how to correlate the events inside same data files or between scan and alert is a good process as well. Sometimes automated scripts/process could help a lot to find some hotspots. For example: in our case, the simple_ip_port_count.pl could quickly identify the evil behavior targeting 53 as well as “Agobot like” behavior.
In many cases, when we drill down to the bottom, we want to see the payload of the event, which could provide more confidence on the final conclusion.

Many thanks to Colby (GCIA 0490), Alastair and XiaoSu who gave a quick review on this paper before submitted.

Appendix

References:

1. Stephen Northcutt, Judy Novak
   <<Network Intrusion Detection>> Third Edition
2. Stephen Northcutt, Mark Cooper, Matt Fearnow, Karen Frederick
   <<Intrusion Signatures and Analysis>>
3. Ed Skoudis <<Counter Hacker – A step-by-step Guide to computer attacks and effective defenses>>
4. Richard Stevens <<TCP/IP illustrated>>
5. Brett Hutley - GCIA practical assignment
   http://www.giac.org/certified_professionals/practicals/gcia/0775.php
6. JUNG — the Java Universal Network/Graph Framework
   http://jung.sourceforge.net
7. WhiteHat Lion Internet worm analysis
   http://www.whitehats.com/library/worms/lion
8. WhiteHat ADM worm analysis
   http://www.whitehats.com/library/worms/adm/
9. Sans advisory about Lion worm http://www.sans.org/y2k/lion.htm
10. LOGS: GIAC GCIA Version 3.3 Practical Detect,
    http://www.dshield.org/pipermail/intrusions/2003-May/007803.php
11. CVE2001-0010
    http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-10
12. CVE1999-0009
    http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-9
13. Thomas Ptacek and Timothy Newsham IDS paper
    http://secinf.net/info/ids/idspaper/idspaper.html
14. Mark Handley, Vern Paxson and Christian Kreibich: "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics" http://www.icir.org/vern/papers/norm-usenix-sec-01-html/
15. Securityfocus: "IDS Evasion Techniques and Tactics".
    http://www.securityfocus.com/printable/infocus/1577
16. Maarten Van Horenbeeck's GCIA paper
    http://www.daemon.be/~maarten/Maarten_Vanhorenbeeck_GCIA.pdf
17. LSASS vulnerability
    http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx
18. CVE CAN-2003-0533
    http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0533
19. Symantec Security Response – backdoor.sdbot
    http://securityresponse.symantec.com/avcenter/venc/data/backdoor.sdbot.html
20. Dug Song: Fragroute homepage
    http://www.monkey.org/~dugsong/fragroute/
21. Rose fragmentation attack
    http://www.securityfocus.com/archive/1/359144/2004-03-29/2004-04-04/0

22. Wouter Clarie GIAC Certified Intrusion Detection Analyst Practical Assignment V4.1
23. Frans J.H. Kollee GCIA Practical Assignment Version 3.1 http://www.giac.org/certified_professionals/practicals/gcia/0563.php
24. Virus info center - Win32.Agobot http://www3.ca.com/securityadvisor/virusinfo/virus.aspx?id=37776
25. Dave's Port list http://lists.gpick.com/portlist/lookup.asp
26. Internet storm center http://isc.sans.org
27. Symantec Internet Threat Report Volume VII, published March 2005
28. Jorge D. Ortiz-Fuentes GCIA practical assignment version4.1 http://www.giac.org/certified_professionals/practicals/gcia/0769.php
29. Malware analysis template http://www.counterhack.net/malware_template.html
30. Snort rules http://www.snort.org/rules
31. Programs for Large Network Analysis http://vlado.fmf.uni-lj.si/pub/networks/pajek/

32. John's post "LOGS: GIAC GCIA Version 3.3 Practical Detect" http://www.dshield.org/pipermail/intrusions/2003-May/007803.php
33. Al Williams discuss about "Possible trojan server activity" and 27374 activity in his GCIA paper. http://www.whitehats.ca/main/members/Herc_Man/Files/Al_Williams_GCIAPractical.pdf
34. Ian mentioned DNS Named Version Bind Attempt in the forum discussion http://www.dshield.org/pipermail/intrusions/2004-July/008200.html
35. Snort rule about "DNS named version attempt" http://www.snort.org/pub-bin/sigs.cgi?sid=1616

Perl scripts:

simple_ip_port_count.pl

```
#!/usr/bin/perl

# This perl refactored based on the apr.pl
# original author: Frans J.H. Kollee  [Reference: 23]
# Refactored by Yuan Fan

use strict 'vars';
use Getopt::Std;
```

```
my %opts;
my %srcip;
my %alertHash={};
my %novelDstPortTable={};
my %novelSrcPortTable={};
my $topN=10;

if ( ! getopts("hs:d:nmtaxy", \%opts)) { # unknown option
print "run\n";
print "options %opts\n";
&printhelp;
exit 1;
}

if ($opts{h}) { # Help requested
&printhelp;
exit 0;
}

if($opts{m})
{
        $topN=$opts{m};
        print "Will use Top $topN\n";
}


if ($#ARGV+1 <1) {
&printhelp;
exit 1;
}

my $infile = $ARGV[0];

main();

sub main
{
        open (thefile, $infile) || die "*** Can not open $infile ***";
        while (<thefile>)
        {
                chomp;
                #check if it is like xxxx -> xxxx like lines, we should change to
\d+\.\d+\.\d+\.\d+ which is more accurate
                if (m/->/i)
                {
                        #print " seeing xxx -> xxx like lines\r";
```

```
                    /(^.*)\s+(.*)\s+(\d+.\d+.\d+.\d+):(\d+) ->
(\d+.\d+.\d+.\d+):(\d+).*/;
                    #$1 date $2 time $3 srcip $4 srcPort  $5 destIP $6 destPort

                    if($novelDstPortTable{$6})
                    {
                            $novelDstPortTable{$6}++;
                    }
                    else
                    {
                            $novelDstPortTable{$6}=1;
                    }

                    if($novelSrcPortTable{$4})
                    {
                            $novelSrcPortTable{$4}++;
                    }
                    else
                    {
                            $novelSrcPortTable{$4}=1;
                    }


                    if ($srcip{$3}) # src-ip already processed
                    {
                            if ($srcip{$3}->{"$5"}) # and with this dest-ip
                            {
                                    $srcip{$3}->{"$5"}->[0]++ ; # Total connects ;

                                    if ($srcip{$3}->{"$5"}->[2]->{"$6"})  # add 1 to
the port
                                    {
                                            $srcip{$3}->{"$5"}->[2]->{"$6"}++;
                                    }
                                    else
                                    {
                                            $srcip{$3}->{"$5"}->[1]++;
                                            $srcip{$3}->{"$5"}->[2]->{"$6"}=1;
                                    }
                            }
                            else  #create entry for dest-ip
                            {
                                    my $rdp = { }; #create empty referenced hash
                                    $rdp->{"$6"}=1;
                                    my $a=[];  #empty array
                                    $a->[0]=1;  #connects to this ip
```

```
                                        $a->[1]=1;   #different ports
                                        $a->[2]=$rdp; #place holder for the ports
                                        $srcip{$3}->{"$5"}=$a;
                                }


                        }
                        else #create empty entry for this src ip
                        {
                                my $rdp = {}; #create empty hash for dst port
                                $rdp->{"$6"}=1;
                                my $a=[];  #empty array
                                $a->[0]=1;  #connects to this ip
                                $a->[1]=1;   #different ports
                                $a->[2]=$rdp; #place holder for the ports
                                my $rd = {}; #create hash for the dest ip.
                                $rd->{"$5"}=$a; #assign the array info
                                $srcip{$3}=$rd; #add this src-ip to hash

                        }

                }
        }
        close(thefile);
        #&printipstatistic() ;
        &printsourceip();
        &reportAlertsTable();
        if($opts{y})
        {
                &reportTopDstPorts();
        }
}

#
sub printhelp {
print "Usage: apr [-h][-n][-t] [-x] [-y] [-s 'IP'][-d 'IP'] <filename> \n\n";
print "\t<filename> is te name of the Snort portscan log-file\n\n";
print "\tFlags:\n";
print "\t-h\tshow this help message\n";
print "\t-a\tDo not display details\n";
print "\t-n\tDisplay totals for each destination-ip\n";
print "\t-t\tDisplay the totals for each destination-ip\n\n";
print "\t-y\tDisplay the top destination port\n\n";
print "\t-x\tDisplay the top src port\n\n";
print "\t-m\t number set the top N number, by default report Top 5, this one is
used with either -x or -y\n\n";
```

```
print "\t-s\t'IP' report on source-ip only\n";
print "\t-d\t'IP' report on destination-ip only\n";
print "\t\t'IP' can be a perl regular expression\n" ;
}

sub printipstatistic {
my $t0 = 0 ; # Number of processed source-ip's
my $t1 = 0 ; # Number of destination ip's
my $t2 = 0 ; # Number of different port
my $t3 = 0 ; # Number of probes
my $portsnum=0; #add all the ports accessed together including single port
times. from src ip to dst ip

# Sorted on Source IP
my @sortkeys = sort {ip2fullip($a) cmp ip2fullip($b)} keys %srcip ;
foreach (@sortkeys) {
$t0++ ;
my $c1 = $_ ; # Column 1
my $r = $srcip{$_} ;
my @sortkeys2 = sort keys %{$r} ;
$t1 = 0 ; # Count the number of source-ip's
$t2 = 0 ; # Number of ports
$t3 = 0 ; # Number of probes
my %prtcnt ; # Create empty named hash
foreach (@sortkeys2) {
$t1++; # Count the destination ip's
my $c2 = $_ ; # Column 2
my $rp = $r->{$_}->[2] ; # Reference to the port hash
my @sortkeys3 = sort keys %{$rp};
foreach (@sortkeys3) {
my $p=$_ ;
$portsnum=$portsnum + $rp->{$_};
if (not $opts{a}) {
print "From $c1 to $c2 on port $p ($rp->{$_} times)\n";
}
if (not exists($prtcnt{"$p"})) {
$prtcnt{"$p"}=0 ;
$t2++ ;
}
$t3 = $t3 + $rp->{$_} ;
}
if ($opts{t}) { # Subtotals per dst-ip
print " $c1 to $c2 are $r->{$_}->[0] connections on" ;
print " $r->{$_}->[1] port(s)\n";
}
}
```

```
if ($opts{n}) { # Subtotals per source-ip
print "IP $c1 probed $t1 destination(s), $t2 distinct port(s) ($t3
alerts)\n";
}
}
print "\n";
}

sub printsourceip {
my $t0 = 0 ; # Number of processed source-ip's
my $t1 = 0 ; # Number of destination ip's
my $t2 = 0 ; # Number of different port
my $t3 = 0 ; # Number of probes
# Sorted on Source IP
my @sortkeys = sort {ip2fullip($a) cmp ip2fullip($b)} keys %srcip ;
foreach (@sortkeys) {
$t0++ ;
my $c1 = $_ ; # Column 1
my $r = $srcip{$_} ;
my @sortkeys2 = sort keys %{$r} ;
$t1 = 0 ; # Count the number of source-ip's
$t2 = 0 ; # Number of ports
$t3 = 0 ; # Number of probes
my %prtcnt ; # Create empty named hash
foreach (@sortkeys2) {
$t1++; # Count the destination ip's
my $c2 = $_ ; # Column 2
my $rp = $r->{$_}->[2] ; # Reference to the port hash
my @sortkeys3 = sort keys %{$rp};
foreach (@sortkeys3) {
my $p=$_ ;


if (not $opts{a}) {
print "From $c1 to $c2 on port $p ($rp->{$_} times)\n";
}
if (not exists($prtcnt{"$p"})) {
$prtcnt{"$p"}=0 ;
$t2++ ;
}
$t3 = $t3 + $rp->{$_} ;
}
if ($opts{t}) { # Subtotals per dst-ip
print " $c1 to $c2 are $r->{$_}->[0] connections on" ;
print " $r->{$_}->[1] port(s)\n";
}
}
if ($opts{n}) { # Subtotals per source-ip
print "IP $c1 probed $t1 destination(s), $t2 distinct port(s) ($t3 alerts)\n";
}
%alertHash->{"$c1"}=$t3;
}
print "\n";
}

sub ip2fullip() { # Needed for sorting the source-ip
my ($ip) = @_ ; my @a = split (/\./, $ip) ;
for (my $i = 0; $i < 4 ; $i++ ) {
while (length($a[$i]) < 3) { $a[$i] = '0'.$a[$i] ;}
}
my $fullip = join('.', @a) ; return $fullip ;
}

#this one count total alerts from one src to a dest ip,
sub totalAlerts()
{
        my $totalNum=0;
        my ($ip) = @_;
        my $ii=0;
        my $r = $srcip{$_->[0]} ;
        my @sortkeys2 = sort keys %{$r} ;
        foreach(@sortkeys2)
        {
                print "+++";
                my $rp = $r->{$_}->[2];
                my @sortkeys= sort keys %{$rp};
                foreach $ii (@sortkeys)
                {
                        #my $p=$_;
                        $totalNum=$totalNum+$rp->{$ii};
                        print "===== $rp->{$ii} =====";
                }
        }
        print "------ From $ip total Alerts is: $totalNum ------\n";
        return $totalNum;

}

#report by the order of the alerts num
sub reportAlertsTable()
{
        my $key;
        my $count=0;
        print "====== Reports sort by Alerts ======\n";
        foreach $key (sort ValueDescendingNum (keys(%alertHash)))
        {
                $count++;
                if($count>$topN)
                {
                        return;
                }

                if(keys(%alertHash))
                {
                        print "======$alertHash{$key}===$key=======\n";
                }
        }
}

sub reportTopDstPorts()
{
        my $key;
        my $count=0;
        print "-------------- Top Destination Ports ---------\n";
#       foreach $key (sort ValueDescendingDstPort (keys(%novelDstPortTable)))
        foreach $key (sort ValueDescendingDstPort (keys(%novelDstPortTable)))
        {
                $count++;
                if($count>$topN)
                {
                        return;
                }

                if($novelDstPortTable{$key})
                {
                        print "-----Port: $key ---> $novelDstPortTable{$key} Times----
-\n";
                }
        }
        print "---------------------------------------------\n";
}

#desending sort
sub ValueDescendingNum {
   $alertHash{$b} <=> $alertHash{$a};
}

sub ValueDescendingDstPort {
  $novelDstPortTable{$b} <=> $novelDstPortTable{$a};
}
```

scanAnalyse2pjnet.pl

```
#!/usr/bin/perl -w
#----------convert scan file to a pajek .net format file for java visualization----------
#----------- Author: Yuan Fan -------------
#-----------    02/12/2005    -------------
#-----------------------------------------

use strict 'vars';
use Getopt::Std;

my %opts;
#this table hold uniqe ip appeared in scan file and the value of each ip is indexed!
#the reason of that is for the pjnet format.
my %iptable;
my $count=1;

my $pjnetStr="*Vertices ";
my $endLine="\n";

my $arclist="*arcslist\n";
my $outputfile="tmp.net";

if ($opts{h}) { # Help requested
&printhelp;
exit 0;
}


if ($#ARGV+1 <1) {
&printhelp;
exit 1;
}

sub printhelp {
print "Usage: scanAnalyse2pjnet [-h] <filename> \n\n";
print "\t<filename> is the name of the Snort portscan log-file\n\n";
}

my $infile = $ARGV[0];
```

```
main();

sub main
{
          open (thefile, $infile) || die "*** Can not open $infile ***";
          while (<thefile>)
          {
                    chomp;
                    #check if it is like xxxx -> xxxx like lines
                    if (m/(^.*)\s+(.*)\s+(\d+.\d+.\d+.\d+):(\d+) ->
(\d+.\d+.\d+.\d+):(\d+).*/i)
                    {
                              /(^.*)\s+(.*)\s+(\d+.\d+.\d+.\d+):(\d+) ->
(\d+.\d+.\d+.\d+):(\d+).*/;
                              #$1 date $2 time $3 iptable $4 srcPort  $5 destIP $6
destPort
                              if (!$iptable{$3})
                              {
                                        $iptable{$3}=$count;
                                        $count=$count+1;
                              }

                              if (!$iptable{$5})
                              {
                                        $iptable{$5}=$count;
                                        $count=$count+1;
                              }

                              $arclist.=$iptable{$3}." ".$iptable{$5}.$endLine;


                    }
          }


          close(thefile);
          #now we construct the $pjnetStr
#scalar keys  $pjnetStr.=
#         &printipstatistic() ;
          &printsourceip();
          print $arclist;
          if ($#ARGV+1 >2) {
                    $outputfile = $ARGV[1];
          }
          #&writeToFile();
}
```

```
sub writeToFile()
{
        open(DATA,">$outputfile");
        print DATA $pjnetStr.$arclist;
        close(DATA);
}


sub printsourceip()
{
        my $key;
        my $count=0;
        my $tmpS="";

        foreach $key (sort ValueDescending (keys(%iptable)))
        {
                $count++;


                if($iptable{$key})
                {
                        $tmpS.=$iptable{$key}." ".$key.$endLine;
                }

        }
        print "#----------------- Auto Generated pjnet file Author: Yuan Fan -------------
------\n";
        $pjnetStr.=$count.$endLine;
        $pjnetStr.=$tmpS;
        print $pjnetStr;
}


sub ValueDescending {
  $iptable{$a} <=> $iptable{$b};
}
```