HTB Retired: Popcorn

Eric Cartman1027

July 7, 2018

1 Disclaimer

This is a walk-through of the tutorial posted by IppSec. This is for INTERNAL USE and PURPOSE OF EDUCATION only. Target machine has ip:10.10.10.6, local ip:10.10.14.10

URL to IppSec's youtube channel can be found here at:
`<https://www.youtube.com/channel/UCa6eh7gCkpPo5XXUDfygQQA>`
Link to this specific tutorial:
`<https://www.youtube.com/watch?v=NMGsnPSm8iw>`

2 Port Scan

Start with scan by nmap, we ran the command: *nmap -sV -sC -oA nmap 10.10.10.6* From the result of

Figure 1: nmap scan.

the scan, we see that port 80 is open for HTTP service. Next, we will set up a proxy using Burp and FireFox to make a HTTP GET Request. And we got a successful response:

Figure 2: Request port80.

3 HTTP Directory Scan

From here, we would like to make a enumeration on hidden/non-hidden directories in the *popcorn* machine. We will do that by running the command:*dirb http://10.10.10.6 -r -o tmp.dirb*. Dirb uses a dictionary(or list of directory's/files) to make HTTP request, and generate a list of directories based on HTTP Response. Here -r is running non-recursive to avoid IDAs, -o means write the output to file tmp.dirb. Note that dirb

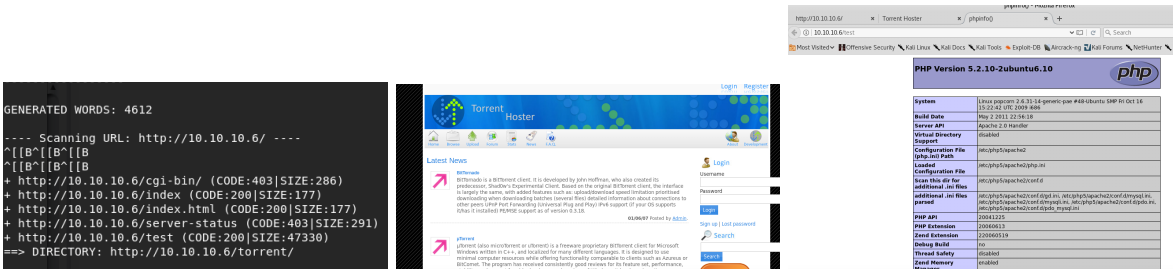

Figure 3: dirb result.

can work with SSL as well(i.e: https). Here, 2 results pop up that can be particularly useful. By clicking onto the link: http://10.10.10.6/test we obtain a php.info() page (2nd pig in fig3). php.info() usually supplies us with about a good amount of information related to the system. Usually phpinfo() is commonly used to check configuration settings and for available predefined variables on a given system. We can identify

The one we are interested in is in DIRECTORY: http://10.10.10.6/torrent/. The last picture in fig3 shows that there exits a torrent server called *Torrent Hoster* in the directory.

From here, we can simply run the command *searchsploit "Torrent Hoster"* to search for any known exploit. Before we devote time into actually implementing the exploit, it's good to **compare the date on which the web server gets configured and the date the exploit was written** to get a general idea about relevancy.

4 Browsing the Torrent Hoster

4.1 Survey

When we survey the Torrent Hoster, we are able to create users, which have the ability to **upload *FILES***.

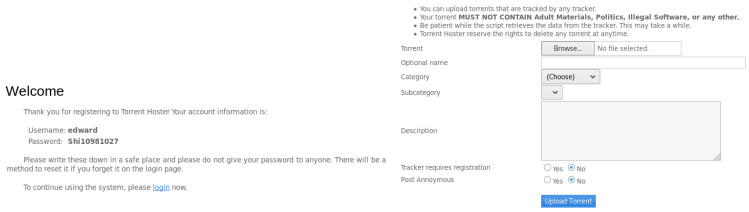

Figure 4: torrent upload

From fig3, we understand that the server has PHP at the backend executing php code. So in order to get a backend shell bound, we will need to upload the php file onto the server using the upload. Notice that to do that, we will need to: and **(1) upload a php scripts that binds a backdoor shell** and **(2) Being able to execute what we uploaded at the backend server**. Note that if we upload some file up from the submission portal, once we click on the link again we will only be able to make a HTTP GET Request and download whatever we just uploaded. In order to actually execute the file at backend, we will have to look into some directories within the server.

4.2 Play with the /torrent/upload directory

Now we know that in order to let what gets uploaded onto the server executed, we will need to get into a directory where we have direct access of all codes. Here by manually trying out a bunch of strings we discover

there is a specific directory within the torrent server called *upload*. The directory is: ***10.10.10.6/torrent/upload***.

Figure 5: /upload

But here, the only thing we see are the png files. Which implies that we are only allowed to access the picture headers we assign to our torrent link. In order to test our assumption, we will upload a torrent file and attach a png as its header, and see if your guess is correct. By uploading a torrent file and uploading a png as its header, we confirm our initial guess:

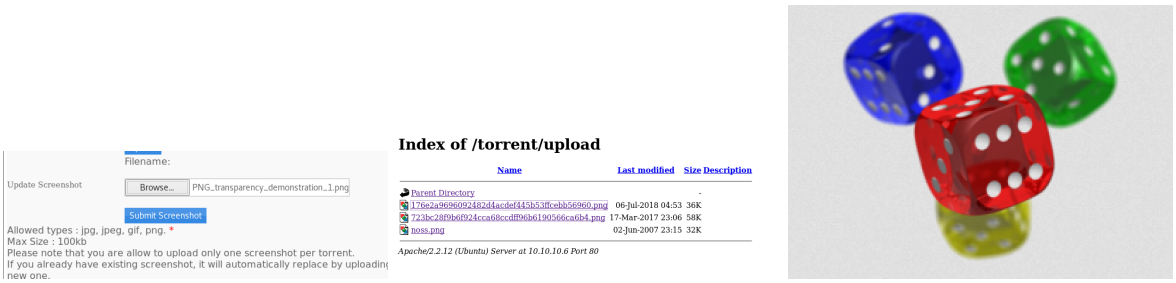

Figure 6: uploadSuccess

4.3 Bypass File Check: Disguise PHP with PNG

Now our goal is clear: upload the php code as **screenshot** onto the server and execute in direcotry**torrent/upload**. However, by simply submit the php file it won't pass the check. Now our goal is to *bypass the file type check on the server's side*. We know from php.info() that the server is a Ubuntu machine, which usually checks file type like png by examining a particular header. Here, we will take a beginning section of the png bytes segmented from Fig6, change that into base64 encoding, and making sure that this section as a header of any file is **sufficient to bypass the file type check**. By encoding the header section in the HTTP Post Request intercepted by Burp in base64, decoding it in Kali Linux, storing it in a file called file and trying to examine its file type, we successfully got the result of a png file. This implies that *Regardless of we append after this heading, the Ubuntu Server machine will always recognize the file containing it as a png file. Hence we will be able to bypass the file check control.*

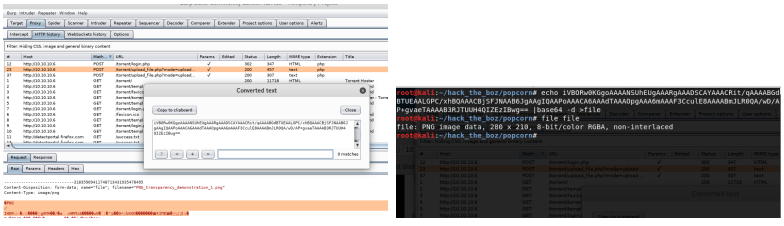

Figure 7: pngHeader

Then, we will use Burp to intercept our HTTP POST Request for the php code, send it to a repeater, and modify the content accordingly. Here are 3 things we mainly modified. The first one is that we prepend

our php code with the section header we just double checked; The second is that we change the Content Type to *image/png*; The last one is that we append a *.png* string within the filename, this together with the first change will tell the Ubuntu Server that this is a valid png file, since by default Ubuntu will judge a file type based on both its section header and a valid "png" string within the filename.

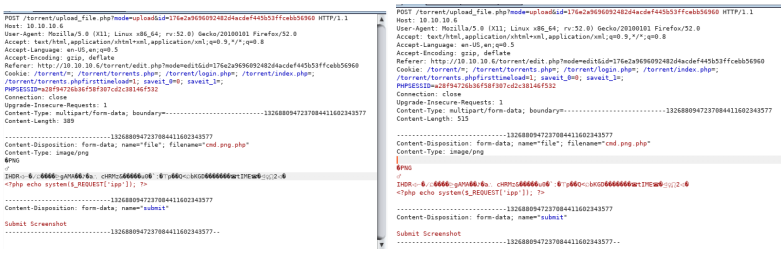

Figure 8: Formating Post Request

As expected, a valid HTTP POST request gets made. The php file got successfully uploaded onto *10.10.10.6/torrent/upload*, and we can see the php file posted on *10.10.10.6/torrent/upload*. See 9

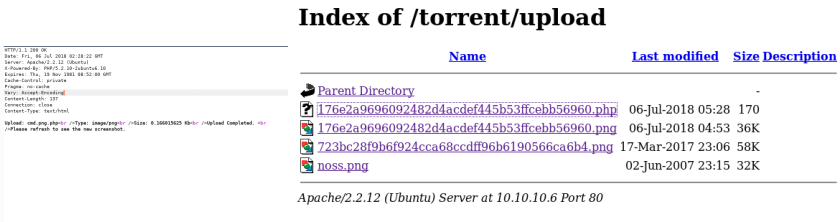

Figure 9: Backdoor Opened

We can now run the command within our php code by entering the command after 'ipp='. Our next step will be to bind a fully functional interactive shell that can allow us to navigate on the server.

4.4 Setting up a Fully Interactive Shell

To begin with, we will set up a HTTP Server using python in our Kali box, and make a HTTP POST and a php execution so that our target machine can connect back to our Kali Box at port 8800 and download the backdoor code from there. 10.

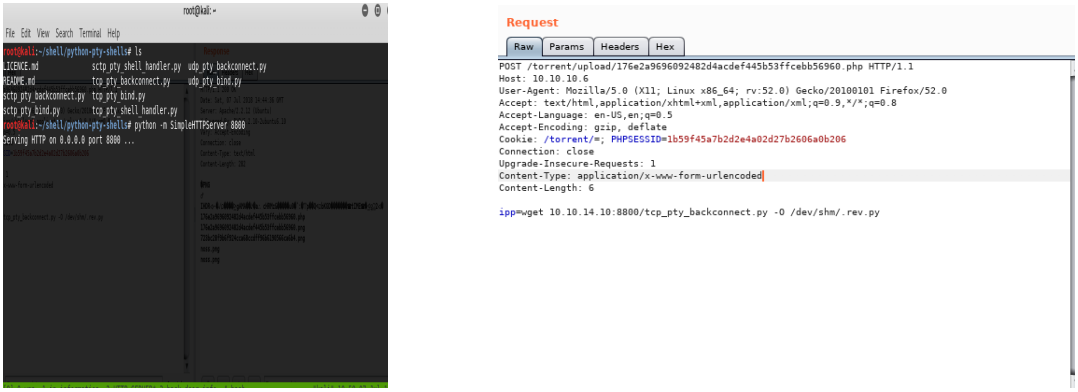

Figure 10: downloading backdoor

After we forward the HTTP POST request via Burp, we can verify that our code gets successfully dropped into the target machine by posting a cat request. Here we confirm that the python file gets dropped correctly at 11

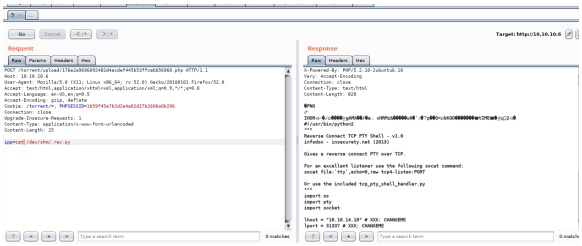

Figure 11: Verifying our download

After checking everything we need to establish a backdoor is in place, we then start listening on port 31337 in our Kali Box, and use Burp to forward another HTTP POST request with which we run the python code we dropped. 12 shows that we correctly establish a reverse shell.

Figure 12: Getting in as www-data

5 Post Exploitation

5.1 First Flag

Very easily we can cd into /home and discover our user flag there. @13

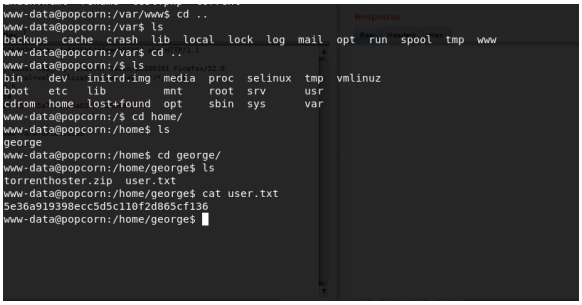

Figure 13: user flag

5.2 Post-exploit Enumeration

5.3 Constructing Appropriate Weapon

After we identified motd as a vulerability, we then use *searhsploit motd* to search for available exploits. Here at the first picture in 14 we see that there is a **MOTD File Tempering Privilege Escalation** we can use. Again before applying this exploit we should double check from the target machine to make sure that version of the target machine ¡ version of the exploit targeted. From the second picture in 14 we confirmed that these versions do match up.

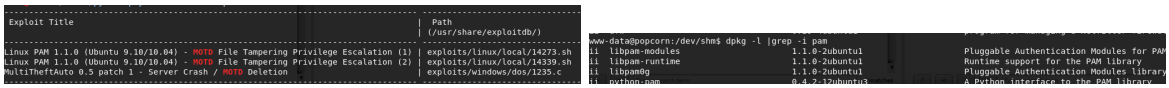

Figure 14: MOTD Exploitation

5.4 Delivering the Payload

After we double check the exploit at 15, we will use the HTTP Server we established before to deliver the payload.

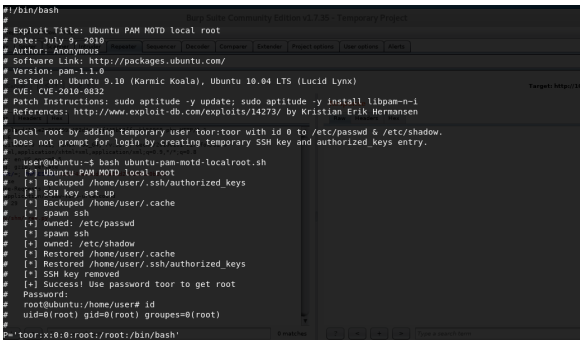

Figure 15: Examining the Payload

After we delivered the payload at 16 , we then run it, and we see that the payload gets successfully executed at 17 , and there by we have the root access.

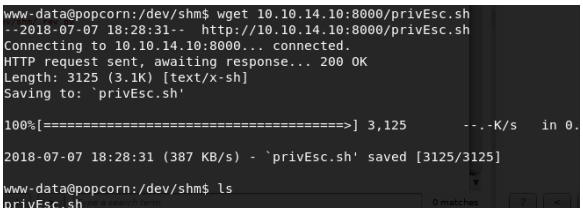

Figure 16: Delivering the payload

Successfully gained root access:

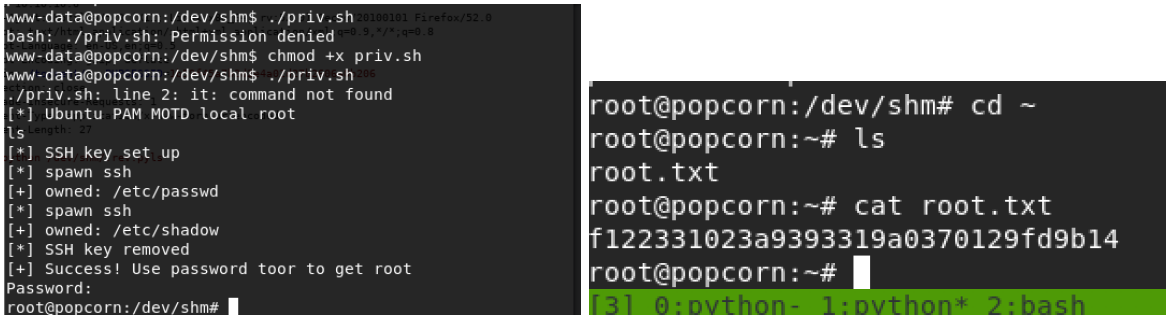

Figure 17: Executing the payload and Finish Collection

6 Conclusion

In conclusion, *popcorn* is a very valuable machine as a first tryout. The most important skill we learn from *popcorn* is the trick to **bypass linux's default check for file type (png) by appending a section of the png header to the payload** and be able to carry out the exploit from there.

See you until next time.

Figure 18: Default Boat.