Gjøvik University College

Ethical Hacking & Penetration Testing

Hack.lu CTF 2014

Assignment #3

Victor Rudolfsson - 120912

October 25, 2014

1 Hack.lu CTF 2014

Alright, so I kind of forgot about this assignment in all the excitement and don't really have any screenshots. Perhaps not that relevant though, considering I only really solved one challenge, although I spent only 7 hours sleeping during the whole time.

I'll first go through the assignment I did solve, and then proceed to the ones I worked on with no success. I'm not at all sure if I can get this down to just 1 page, considering I have to describe the task, how I (we) solved it, and what we attempted. I hope 1 page isn't the maximum limit?

1.1 Solved: Killy the Bit

URL: https://wildwildweb.fluxfingers.net/challenges/13

Killy the Bit was a challenge under the **Web** category, which required some SQL-injection to acquire the flag. The site allowed one to enter ones username, and it would then generate a new password for the user with that username.

However, the input was not sanitized, making it open to injections. We were provided with some PHP code for the site which made it clear that there was only **one** place in the code where anything at all was printed from the database, and this was the *name* column from the *user* table.

The first query looked like this:

Listing 1: Killy the Bit - First Query

```
SELECT name,email FROM user where name='".\$\_GET['name']."'
```

Some PHP code following this made it clear that if this query succeeded to return any rows, then a new password would be generated and the script would end with a die(). However, if it found no rows, it would continue with the next query:

Listing 2: Killy the Bit - Second Query

```
SELECT name,email FROM user where name sounds like '".\$\_GET['name']."'
```

If this query failed, it would just give you an error message. But, if it passed, it would print the *name* column of all returned rows. This gave away quite some hints – first, that the first query must fail but the second one must pass. There's a clear difference between the two, and it's the 'LIKE' keyword in the second one rather than the equal sign in the first one. Second, that any row aliased 'name' would be printed if the second query passed.

I spent quite some hours on this – far more than was necessary in the end, but I figured out that the database had plenty of rows and I could base my query around that, as well as a selective UNION SELECT.

Eventually, I crafted this input for the input field, which returned the flag:

Listing 3: Killy the Bit - My solution

```
adm%n' UNION SELECT
IF(ROW_COUNT()<1, (SELECT passwd as email FROM user WHERE name='admin'),0)
as name, email FROM user LIMIT 126,127 #
```

If you imagine this inserted in the first query, it should not pass mainly because there will not be that many rows returned. The second query will pass, however, because now there'll be enough rows returned for it to find anything with the LIMIT statement. Every single row has the flag as name in the UNION query, so any row from there will return the flag.

1.2 Solved: ImageUpload

URL: https://wildwildweb.fluxfingers.net/challenges/9

I worked on this assignment for an hour or two when I took a break from frustration over Killy the Bit, but I was determined it had something to do with the old PHP resize bug in ImageMagick and attempted every possible way or injecting PHP code into the EXIF parameters. I never got this to work and went back to work on Killy the Bit, which I did manage to solve.

Eventually, I went back to work on this but now Tommy who was sitting next to me had gotten quite far on it and managed to get some basic SQL injection working with the EXIF fields.

The challenge was to upload an image to the site, and get the flag. You could log in, and the usernames to log in with were either Sheriff or Deputy. The web page listed all images you had uploaded recently, along with the fields *Manufacturer*, *Model*, *Author*, *Width* and *Height*. Width and Height were calculated whereas the others are exif fields, namely *artist*, *model*, *make*.

In any of these fields we could get an injection by surrounding it in '+(QUERY HERE)+', but for the most part, we couldn't print everything.

As we found out from talking to people after we had solved it, this whole challenge could be solved with a single query. However, we resorted to much more extreme measures.

We managed to get the password for the user *deputy*, but when logged in it just told us that only *sheriff* can see the flag. Whenever we tried to get Sheriff's password, we just got *414*. We could for some reason not print a lot of text, especially not whatever we found in the database, and often had to use HEX() functions in MySQL and get the data as hex, then convert it afterwards into ASCII.

After hours of frustration trying to get the password for sheriff, we started picking out the password character by character (something we should have done a lot earlier).

Eventually, we had the password *AO7eikkOcucCFJOyyaaQ* and this was where we got really, really frustrated. We finally had the password, after a long time of..

Listing 4: ImageUpload - Crafting an image

```
exiftool -model="" -artist="'++'" \\
-make="'+(SELECT HEX(SUBSTR(password),3,1) FROM users WHERE id=1)+'" photo.jpg
```

..followed by an upload and decoding of HEX (and for those values that wouldnt work in hex, ASCII()). We now had the password but we still couldn't log in with it and we could find nothing that was wrong. We considered even such a stupid thing that what if they had hardcoded a special hash comparison for JUST that user account? Because the user "deputy" could be logged in with the password we found for it.

Eventually, we went home and went to bed as it was already 5 AM. The following day I kept working on this, now taking a different approach: I wanted to get a shell. I tried

to get all the table names out of the database, and found that it contained:

1. **users(***(id, name, password)*);
2. **pictures(***id, path, width, height, Author, Manufacturer, Model*); and
3. **brute(***ip, time*)

I thought if I could get the path for one of the images and change this path, I would be a step in the right direction at least. It took me quite some time working on this, when suddenly Tommy walks into the room and declares he has solved it and turned the flag in: The solution was that **one** of the letters in the password we retrieved was the wrong case. The first C should have been uppercase. And that allowed us to log in and take the flag.

Oh, and the 414 that we kept receiving meant "END OF TRANSMISSION" which was the reason the hex value would just stop there when we tried to get the whole password as hex.

1.3 Solved: FluxSaloon

URL: https://wildwildweb.fluxfingers.net/challenges/35

This was an easy one. The challenge was to chug 1 bottle of beer (50cl), film it, and make sure to show that the bottle was unopened, contained alcohol, and that it was empty by the end. We also had to show our team name, and some proof that it was an up-to-date video.

The points would be awarded as 100 points minus the amount of seconds it would take you to chug the whole beer; with 0-35 bonus points awarded if you made the judges laugh somehow.

As I felt I needed to take a break and focus on something else for a while, I decided to take all the empty pizza boxes, and rebuild them into a robot armor, with a sheriff hat (combining last years Hack.lu theme of Robots with this years theme of the Wild West).

The armor consisted of a chestplate, backplate, arms, legs, tabard, shoulder-pads, helmet and a sheriff hat. All made out of Peppe's Pizza boxes.

Jan, our German beer chugging champion, took on the task of chugging the beer and wearing the armor. I actually think we were among the highest scoring teams on this task, as we got a whopping 104 points! It took Jan approximately 21 seconds to chug the beer, giving us 25 out of 35 possible bonus points!

1.4 Dalton's Corporate Security Safe for Business

URL: https://wildwildweb.fluxfingers.net/challenges/18

This challenge presented us with a page where you had to enter numbers displayed in an HTML5 canvas element. You had to do this over and over, and every time the time you had to do it was shorter than the last. First, I tried just entering it as fast as I could, relying on my typing speed and eyesight. This, of course, failed.

Eventually, I thought I could write a Ruby script that would fetch the site, figure out the variables for the numbers displayed in the javascript, enter it, and submit it, over and over again. I wrote a long case/when statement in Ruby to go through the javascript line by line, and populate an array with the *variable names* in JavaScript. Then I tried to verify that these were the correct ones in the correct order, and this is about as far as I came in my Ruby script before *Mirek* raises his arms in celebration declaring he solved the challenge with a Greasemonkey script.

1.5 Next Global Backdoor

URL: https://wildwildweb.fluxfingers.net/challenges/30

This task is the most amazing one I came across and the one I am most pissed off at not managing. We were given a URL and a PHP script as such:

Listing 5: Next Global Backdoor

```
<body bgcolor="black">
```

```
<center><img src="img.png"></center>
<?php
@\$GLOBALS
=\$GLOBALS{next}
=next(\$GLOBALS{'GLOBALS'})[\$GLOBALS['next']['next']
=next(\$GLOBALS)['GLOBALS']][\$next['GLOBALS']
=next(\$GLOBALS[GLOBALS]['GLOBALS'])[\$next['next']]][\$next['GLOBALS']
=next(\$next['GLOBALS'])][\$GLOBALS[next]['next'](\$GLOBALS['next']{'GLOBALS'})
=next(neXt(\${'next'}['next']));
?>
```

I started following the code, operation by operation, but quickly got lost and had to start over. Recursion in recursion in recursion .... My initial idea after 20min of going through the code was that it pointed all fields in the $GLOBALS array to one field, and executed a command from this one. I had a hunch that it would be the $_COOKIE or $_POST fields, so after a while I tried most combinations I thought it might be with a Chromium extension (POSTman), posting data to *globals*, *next*, and so on, while running the script locally to be able to read the logs at the same time. I tried the $_COOKIE['next'] as well, with no success (THIS pissed me off when I found the answer in the write-ups!).

Nothing worked. Me and Tommy began working on it together, debugging the PHP code with php-dbg, using a REPL / IRB program for PHP called Boris, and tinkering with the code. Still, no success.

We spent many many hours on this one, with no success. When the CTF was over and the write-ups came out, we found that it executes a command from $_COOKIE['GLOBAL'] (come on, the whole code is TWO words, GLOBAL and next, and we tried $_COOKIE['next'] and forgot about GLOBAL?! God effing damnit.) with an argument from $_GLOBAL[$_COOKIE['GLOBAL']]['type']

1.6 HotCows Dating

URL: https://wildwildweb.fluxfingers.net/challenges/21

This challenge was hilarious. We were presented with a dating site where you could live chat with cows, who would occasionally **moo seductively** . If you wrote anything containing *flag* to them, they'd inform you that you needed a PREMIUM account.

We could set the *has_premium* flag locally in JavaScript, but it changed nothing. Except now Abbie the hottest cow in all the lands would reply that the secret is that the *premium_id* is actually the flag.

After checking robots.txt and realizing *.git* was a disallowed directory, we managed to get the git repo from the site along with all source code. This source code indicated that the flag was displayed on top of the chat for all users with either the *support* or *premium* field set to true.

It brought us to the conclusion that we had to use the only function available on the site: Report problem. If we could report a problem, the support agent would read it. If we then could get some XSS into the site that would somehow steal the flag, and then post it to our server, we would have it!

We spent SO many hours working on crafting an XSS that wouldn't be blocked by the Content Security Policy on the site, to no success, and as much as I tried to get a joint operation on this challenge during the last 4 hours of the CTF ... we got nowhere with it.