Summer Challenge - Writeup

@ArmySick / clueless

September 2018

1 Challenge 1 - Have a l00k (recon)

Knowing this is a recon challenge, after some exploring of the webpage, checking the source codes for any easy flags or any hidden/invisible text or form on the page, it was time to scan for some directories that the server shouldn't be revealing.

Let's fire up dirsearch [1] and see if there is something interesting we can find.

```
> python3 dirsearch.py -u <page_URL> -e asp,php,html,txt

-u receives the page URL (do not include the index.html on it).
-e specifies what file extensions to search for.
```

Although the output was somewhat verbose, it didn't take long to find some potentially interesting files and folders.

The most common (and relevant) occurrence was files within the .git folder that returned 200 status code (Server Ok), so they are accessible.

So, the next logical step was to somehow do a sort of a git clone to my local machine for better inspection.

```
> wget -r --no-parent <page_url>/.git/

-r does a recursive wget
--no-parent doesn't download parent folder
    (just .git folder recursively is enough).
```

During some of the exploring of the directory and its sub-directories, the idea came up that since the folder is a .git folder, git commands can probably be used upon it. After pondering some git reset or checkouts commands, and even checking to see if the git repository on GitHub was public, the commit history logs revealed the place where the flag could be:

```
> git log
```

[1] https://github.com/maurosoria/dirsearch

```
commit f2c9fd9b892a806afb018c391e15d7221c613773
Author: Ricardo <ricardojba@users.noreply.github.com>
Date:   Fri Sep 29 08:57:40 2017 +0100

    Update 8JUCv3fZ44.html

(...)
```

Let's check it! CHALLENGE_URL/8JUCv3fZ44.html :

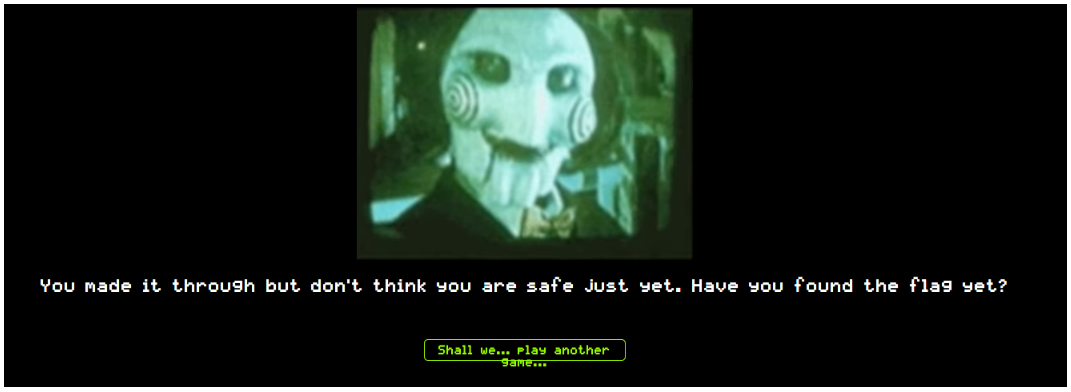

You made it through but don't think you are safe just yet. Have you found the flag yet?

Figure 1: Challenge 1 done!

Nice! We got the flag on the page's source code.

2 Challenge 2 - Lost in Translation (misc)

Okay, I'll admit it. I went too deep in this challenge.

Thoughts way too complex came to mind after reading the description of the challenge mentioning Chinese/Mandarin and seeing the message:

```
-[------->+<]>---.--[--->+<]>.-----------.++++++.+ (...)
```

After more time than I want to admit dabbling around with character encodings and trying to learn chinese representation of characters bit/byte-wise and how to convert and handle them in python, a sudden thought popped into my head and reminded of me of this crazy programming language "brainfuck".

Wikipedia confirmed that the characters used to code in this language were very similar, so after finding an online compiler, a message with the flag popped up.

3 Challenge 3 - Crack me if you can! (reverse)

After heading to the challenge page, a binary is downloadable.

Let's start it up by inspecting the file. It is confirmed to be an ELF 64-bit executable.

Let's run it! It asks for a password. Trying a lucky guess or sending no password results in "Login failed!".

Digging deeper, we can ltrace the executable to see what is going on behind the scenes.

```
> ltrace ./crackmeifyoucan

__libc_start_main(0x4005d6, 1, 0x7ffc4543b4f8, 0x400770 <unfinished ...>
printf("\nEnter the password: "
)
gets(0x7ffc4543b3c0, 0x23ce010, 0x7fc1338df780, 21Enter the password: abc
)
memcmp(0x7ffc4543b3c0, 0x7ffc4543b3e0, 21, 0x7ffc4543b3e0)
puts("\nChecking password...\n"
Checking password...

)
puts("Login failed!\n"Login failed!

)
```

Oh look a 'gets' function! Seems like great news for us, bad news for the developer of the executable. Can we overflow that?

Looking further, the 'memcmp' function compares the first 21 bytes of the address where 'gets' reads into (0x7ffc4543b3c0 in the example) with the first 21 bytes of 32 addresses above (0x7ffc4543b3e0). No matter how many executions are made, the address will change but the address it is compared to will always be the first address + 32.

Putting the puzzle pieces together, we could maybe use the insecure gets function to insert some 21 bytes into the the first address and overflow that, guaranteeing that, starting 32 bytes after, the next 21 bytes will be equal to the first.

While I did explore a bit of gdb to make sure that the buffer overflow was going to the correct places and that this theory could work, it was just an intermediate step. The tl;dr is:

Let *SAME_STR* be a string of length 21. The password must be:

SAME_STR + 11 random characters + SAME_STR

All characters must be 1 byte long.

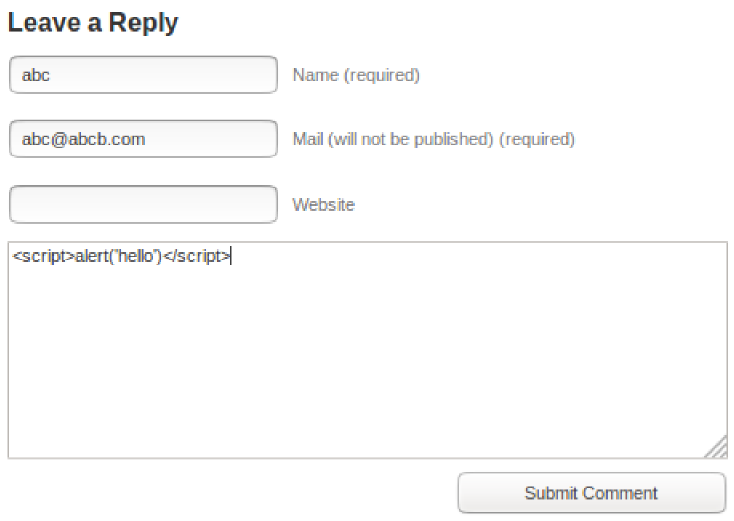
We can add some few extra bytes to the input password, but if it goes too long, segmentation fault will occur without revealing the flag and the next url.

Great! Onto the next challenge.

4 Challenge 4 - We love Web 2.0 (web)

Seeing as this page is a WordPress one, let's fire up a wpscan [2] while we explore the page a bit.

There is a comment section, so how about we try some XSS while we wait?



abc

abc@abcb.com

```
<script>alert('hello')</script>
```

Figure 2: Comments XSS

Hmm.. It didn't work. Maybe we could have insisted a bit more on it, but wpscan is already over and seems to have some information for us. Apparently, the version of WordPress is 2.3.1 which was released in 2007.

Among others, there is a very useful identified vulnerability in this version of WordPress. It is a Charset Remote SQL Injection and — conveniently — wpscan provides us with a ExploitDB link that contains a proof of concept, https://www.exploit-db.com/exploits/4721/. The proof of concept says we can perform SQL injection by accessing the following URL:

```
<CHALLENGE_URL>/?exact=1&sentence=1&s=%b3%27)))/**/AND/**/ID=-1/**
/UNION/**/SELECT/**/1,2,3,4,5,user_pass,7,8,9,10,11,12,13,14,15,16,
17,18,19,20,21,22,23,24/**/FROM/**/wp_users%23
```

The URL above allows us to see the user_pass of the admin user, as confirmed by the schema of the WordPress database (https://codex.wordpress.org/Database_Description).

'ccf5538dc31d435d6bab145c924041d8' hmmm what an unusual password. 32 character long alphanumeric? Hopefully it is a MD5 hash. Using a random MD5 decrypter online, we finally got our admin password!

After logging to the administrator dashboard and doing some exploring (maybe there's an hidden flag here?), I decided to try and use the activated

[2] https://wpscan.org/

”Hello, Dolly” plugin to my advantage and edit the php out to get me a reverse shell. After some manual testing to confirm that this was possible, I uploaded a web shell created by weevely [3]. A simple ’ls’ on the challenge page’s directory revealed a .txt with the flag. Awesome!

[3]https://github.com/epinna/weevely3

5 Challenge 5 - Explorer (web)

We're in. But our user (www-data) is still limited to what it can do inside the machine. After some standard exploring (checking folders, confirming no access to root, confirming that we're not inside sudoers list, checking /etc/passwd, ...) we've finally rang a bell: There is an executable with SUID permission set, named "SecOps-Backup-Service", which means it is ran as root. It looks very promising.

Executing it, we get the output:

```
--=Super Secure SCP Backup Service=--

Backing up your data with scp now...
```

Oh, the irony! This pretty much confirms this could be a door. Noticing that the program uses scp, initially I thought of editing the PATH variable in a way that would execute a custom made 'scp'-named program instead of the actual expected scp. Unfortunately, I changed the PATH variable in one command and THEN ran the executable.

This made me explore other solutions [skip this paragraph if you don't care about the wasted attempts]. Started by exploring the executable and finding out that the scp command it was executing was "scp root@secops-srv:data.tgz .". Maybe there was a way we could set a controlled machine as "secops-srv"? After attemtping to mess with /etc/hosts and /etc/resolv.conf files, I just let the idea go, as even if I was able to insert some of my code into the folder as root, it would be a bit complex to get some code that actually does anything relevant. Seemed a bit too farfetched.

So, while trying an approach on the network (maybe it was needed to get control of another machine on the machine's network) and exploring the 'get-ip-address' executable in the same folder as the SecOps-Backup-Service executable, I had an illumination that you could indeed edit the PATH variable, just gotta do it inline with the ELF's execution (doh!). Hence, I set the PATH variable to /tmp/ + the original PATH, and just created an executable named "scp" in /tmp/ that would execute my code as root.

```
> cat /tmp/scp
whoami

> PATH=/tmp/:$PATH /usr/local/bin/SecOps-Backup-Service
root

--=Super Secure SCP Backup Service=--

Backing up your data with scp now...
```

Fantastic! We have now root privileges.

After some browsing, the flag revealed itself to be inside user vibrio's home folder.

6 Challenge 6 - Climbing to the Moon (pwn)

Maybe we went too far in Challenge 5. Because of the used approach, this was as simple as looking inside /root/ folder to find the final flag. Since root privileges were already obtained, it wasn't a problem.

Can't wait to read other people's approaches on Challenge 5 because I — probably — failed to find the expected attack vector for Challenge 5.

7 Conclusions

Great fun and it was a great recon to pwn experience (except challenge 2. Let's not talk about challenge 2).

The challenges were all doable without any advanced tools or equipment, while keeping an intuitive flow.

Thank you @vibrio and @simps0n for providing an amazing and fun opportunity to further acquire and consolidate my knowledge.