CS177 Computer Security Discussion
Spring 2020 - Week 7

Dongyu (Hector) Meng
May 11th
Today

- `minecraft_hello` and `minecraft` (more on `lazy_panel` next week)
- Challenge roadmap
- Know your tools
  - Pwntools (connection, shellcoding)
  - Ghidra
  - GDB
minecraft_hello

- Run it
- How the service is being run on the server?
- Take a look at the source
- Compilation options and **checksec**
- *mine* leaks the values on stack
- Overflow *buf* when you leave your name *carefully* (more later)
- Jump to win to get a shell
- Cat the flag
Talk to the program

- Exploit locally before you try remote exploit!
- Talk to the program with **pwntools**

```python
from pwn import *
context(arch = 'i386', os = 'linux')
context.log_level = "DEBUG"

r = remote('exploitme.example.com', 31337)
# r = process('./your_binary')
r.send(asm(shellcraft.sh()))
r.interactive()
```
Dig the binary

- Source code is not low-level enough!
- Compiler makes a lot of decisions
- Inspect the binary with **Ghidra** (IDA, r2, Binary Ninja)
- Get some intuitions to map things together (run helper)
Things go wrong

- Debug!
- **GDB** is your friend.
- Or just use **attach** with pwntools to attach to a running process
- Example
The catch

- In the epilog, the challenge binary does something different from the calling convention introduced in class
- Return address is not a special place on memory!
- *ret* jumps to the address right on top of the stack and pops
- Solve that and you get the challenge
 Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td></td>
</tr>
<tr>
<td>%rbx</td>
<td></td>
</tr>
<tr>
<td>%rcx</td>
<td>0x4</td>
</tr>
<tr>
<td>%rdx</td>
<td>0x100</td>
</tr>
<tr>
<td>%rdi</td>
<td></td>
</tr>
<tr>
<td>%rsi</td>
<td></td>
</tr>
</tbody>
</table>

 Memory

<table>
<thead>
<tr>
<th>Word Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x400</td>
</tr>
<tr>
<td>0xF</td>
</tr>
<tr>
<td>0x8</td>
</tr>
<tr>
<td>0x10</td>
</tr>
<tr>
<td>0x1</td>
</tr>
</tbody>
</table>

 Assembly Code:

```
leaq (%rdx,%rcx,4), %rax
movq (%rdx,%rcx,4), %rbx
leaq (%rdx), %rdi
movq (%rdx), %rsi
```
<table>
<thead>
<tr>
<th>Registers</th>
<th>Memory</th>
<th>Word Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax 0x110</td>
<td>0x400</td>
<td>0x120</td>
</tr>
<tr>
<td>%rbx 0x8</td>
<td>0xF</td>
<td>0x118</td>
</tr>
<tr>
<td>%rcx 0x4</td>
<td>0x8</td>
<td>0x110</td>
</tr>
<tr>
<td>%rdx 0x100</td>
<td>0x10</td>
<td>0x108</td>
</tr>
<tr>
<td>%rdi 0x100</td>
<td></td>
<td>0x100</td>
</tr>
<tr>
<td>%rsi 0x1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

leaq (%rdx, %rcx, 4), %rax
movq (%rdx, %rcx, 4), %rbx
leaq (%rdx), %rdi
movq (%rdx), %rsi
- *mine* leaks the values on stack
- Find out where the stack is as well as the canary with *mine*
- Overflow *buf* when you leave your name *carefully*
- Leave the shellcode you crafted at the same time
- Jump to shellcode to get a shell
- Cat the flag
Shellcoding with Pwntools

Pwntools shellcraft