Advanced Cybersecurity Topics

–

Binary Mitigations and how to bypass them

19-20

Binary defence mechanism!

- Stack Canary
- Address Space Layout Randomization
- Position Independent Executable
- Not eXecutable bit
- RELocation Read Only

Stack Canary

- **Compiler** insert a pseudo-random value in the stack between local variables and return address
- The value is checked at the **return**
- If the value changed something bad happened. (**Abort**)

Example: gcc -fstack-protector

```
0804844b <vuln>:
 804844b:       55                      push   %el
 804844c:       89 e5                   mov    %es
 804844e:       83 ec 18                sub    $0x18,%esp
 8048451:       65 a1 14 00 00 00       mov    %gs:0x14,%eax
 8048457:       89 45 fc                mov    %eax,-0x4(%ebp)
 804845a:       31 c0                   xor    %eax,%eax
 804845c:       8d 45 e8                lea    -0x18(%ebp)
 804845f:       50                      push   %eax
 8048460:       e8 ab fe ff ff          call   8048310 <gets@plt>
 8048465:       83 c4 04                add    $0x4,%esp
 8048468:       8b 55 fc                mov    -0x4(%ebp),%edx
 804846b:       65 33 15 14 00 00 00    xor    %gs:0x14,%edx
 8048472:       74 05                   je     8048479 <vuln+0x2e>
 8048474:       e8 a7 fe ff ff          call   8048320 <__stack_chk_fail@plt>
 8048479:       c9                      leave
 804847a:       c3                      ret
```

%gs:0x14 contains the canary, initialized by the kernel with a random value when the process starts

If canary is tampered with, **abort** (without returning)

Bypass Stack Canary

- **Not overwrite** the canary!
  - Need a vulnerability that lets you write saved EIP and not the canary.
- Overwrite the canary with the **right value**
  - Need a memory leak that lets you read the canary
- Overwrite **error handling** function.
  - <__stack_chk_fail@**plt**>

Address Space Layout Randomization

- **Randomize Base** Address of Sections
- Enabled into the **kernel**, enforced by the loader
- Stack, Libraries and Heap can always be randomized
- .text is randomized only if the binary is compiled as **Position Independent Executable** (PIE)

ASLR Attacks

- .text Section (and .got, .bss, .rodata etc.) are **not** always **randomized**.
- Randomization works per page (4k bytes):
  - **Leak** an address and you know all the addresses
  - **Contiguous pages** stay contiguous
    - Leak a .bss address and you know .text and .got
- **Probabilistic attack**: you can overwrite the first 2 bytes and get the right value with 6.25% probability (~50% 10 attempt, >95% 50 attempts)
- Side Channel Attacks: ASLR on the Line: Practical Cache Attacks on the MMU

Interesting Stuff and where to Leak them

LibC:	stack (return address), got, main_arena
.**text** (when PIE):	stack, function pointers
Canary:	stack
Stack:	stack (stack frames)
Heap:	stack, heap (pointers)

Not eXecutable bit

- Pages have **permissions**
- If X bit is not set (or NX is set) **processor** will not execute the page
- only .text is executable (not really true)
- you do **not** have a **WX page**.

NX Bypass

- Try to execute code that's **already executable**
- Write inside the **GOT**
  - you can change function pointers.
  - There always be there some function pointer in GOT
- Create a **RWX** page
  - you may try to trigger a memprotect
  - using ret to libc
- **ROP**

GOT Protection -RELRO

- **Partial RELRO:** Do not put .got after .bss and avoid overflow from bss
- **Full RELRO**: Do not be lazy
  - Load everything at the beginning, make .got Read Only

RELRO Bypass

- Partial RELRO:
  - Any **arbitrary write**.
- Full RELRO:
  - You can still overwrite the saved EIP
    - Ret to LIBC
    - **ROP**
  - Leakless (How the ELF Ruined Christmas)
- .got it is always a good point to search for a **leak of libc** (or any other lib)