Lec05: Bypassing DEP with Return-Oriented Programming

Chengyu Song

Slides based on CS6265 taught by Prof. Taesoo Kim
Scoreboard
Administrative

- Due: Lab04 is extended to **Feb 20** at midnight
- Due: Lab05 is out and its due on **Feb 23** at midnight (extendable)
- 7 normal challenges + 3 bonus challenges
- Survey: kernel or web?
Lab04:

- xor, weak-random: 7-8
- terminator, assassination, stackshield, mini-heartbleed, ssp: 5-4
- gs-random, pltgot, fd: 3
Discussion: Lab04

- What's the most "interesting" bug or challenge?
- So, should we use canary or not?
- So, which one would you like to use?
Reminder: Subtle Design Choices

• Where to put the canary? (e.g., right above RA? fp? local vars?)
• Which value should I use as a canary? (e.g., secrete? random? per exec? per func?)
• How to compare the canary value? (e.g., xor? cmp?)
• What to do after you find the canary value is corrupted? (e.g., crash? report?)
Discussion: Weakness in Lab04

- xor: static canary
- weak-random: dynamic but guessable
- terminator: frame pointer not protected
- stackshield: no integrity check + funcptr
- assassination: important local variable not protected
Discussion: Weakness in Lab04

- mini-heartbleed: secret leaked
- gs-random: argument → arbitrary write
- pltgot: funcptr + failure handling
- ssp: argument + failure handling
- fd: vtable ptr
Discussion: fd

- Overwriting 'struct FILE'
Take-outs from Stack Canary?

- Stack Canary indirectly protects the "integrity" of RA, funcptr, etc
  - (e.g., exploitation mitigation $\rightarrow$ NX, canary)
- We better prevent buffer overflows at the first place
  - (e.g., code analysis, better APIs)
Discussion: How to make exploitation more difficult?

- What if the stack/heap memory is not executable?
  - Then, where to put your shellcode?
- What if the stack address (or code/heap) is random?
  - Could you exploit any challenge in the last week?
DEP: data execution prevention

- Goal: make data not executable
- Hardware support
  - NX, XN: control whether a page is executable or not
  - WXN: automatic W^X
void start() {
    printf("IOLI Crackme Level 0x00\n");
    printf("Password: ");
    char buf[32];
    memset(buf, 0, sizeof(buf));
    read(0, buf, 256);
    if (!strcmp(buf, "250382"))
        printf("Password OK :)\n");
    else
        printf("Invalid Password!\n");
}
DEP in Action

$ cc -m32 -g -O0 -fno-stack-protector -o crackme0x00-nossp-noexec crackme0x00.c
$ bin/checksec.sh --file crackme0x00-nossp-noexec

RELRO    STACK CANARY   NX      PIE      RPATH
Partial RELRO No canary found NX enabled PIE enabled No
-------------------- -----------
DEP in Action

$ ./crackme0x00-nossp-exec < input
IOLI Crackme Level 0x00
Password:Invalid Password!
== FLAG ==

$ ./crackme0x00-nossp-noexec < input
IOLI Crackme Level 0x00
Password:Invalid Password!
Segmentation fault
Code Reuse Attacks

- If we cannot inject code, can't we reuse existing ones?
  - Lab03: simple-bof
  - Lab04: stackshield
  - Lab04: pltgot
Code Reuse Attacks: Timeline

- Return-into-libc
- Return-Oriented Programming
  - Turing complete!!
- Fine-grained randomization
  - Just-in-time Code Reuse
- Control-flow Integrity
  - Out-of-Control: Overcoming Control-Flow Integrity
  - Counterfeit Object-Oriented Programming (COOP)
  - Control Jujutsu
Today's Tutorial

• In-class tutorial:
  • Ret-to-libc
  • Code pointer leakage / gadget finding
  • First ROP!
int main(int argc, char *argv[])
{
    setvbuf(stdout, NULL, _IONBF, 0);
    setvbuf(stdin, NULL, _IONBF, 0);

    void *self = dlopen(NULL, RTLD_NOW);
    printf("stack   : %p\n", &argc);
    printf("printf(): %p\n", dlsym(self, "printf"));

    start();

    return 0;
}
Ret-to-libc: printf

[buf ]
[.....]
[ra ] -> printf
[dummy]
[arg1 ] -> "Password OK :)


Ret-to-libc: system

[buf]  
[..]  
[ra] -> system  
[dummy]  
[arg1] -> "/bin/sh"
Chaining Two Function Calls

printf("Password OK:")
system("/bin/sh")
Chaining Two Function Calls

[buf]  
[.....]  
[old-ra] -> 1) printf  
[ra] --------------> 2) system  
[old-arg1] -> 1) "Password OK :)
[arg1] -> "/bin/sh"
Chaining N Function Calls

[buf    ]
[.....  ]
[old-ra ] -> 1) printf
[ra     ] -------------------> pop/ret gadget
[old-arg1] -> 1) "Password OK :)
[ra     ] -> 2) system
[ra     ] -------------------> pop/ret gadget
[arg1   ] -> "/bin/sh"
[ra     ] ...
Tutorial Goal: Chaining Three Calls

printf("Password OK:\")

system("/bin/sh")

exit(0)
References

- ROP