Lab 3: Binary-Level CFI Implementation

Objective

The objective of this lab is to implement simple CFI to directly harden binary code.

Test Programs

Two test programs are given here: https://tinyurl.com/3xtzkzzy

There is an exploit input for each of them to hijack the control flow. Run them in Linux:

./ex01 < exploit_ex01.bin. #hijack the return address

./bof3 < exploit_bof3.bin #hijack a function pointer on the stack

Binary Rewriting

We will use datalog disassembly: https://github.com/GrammaTech/ddisasm

You can directly use its docker image:

```
docker pull grammatech/ddisasm:latest
docker run -v "`pwd`":/shared -it grammatech/ddisasm bash
cd /shared
ddisasm <test_binary> --asm <test_binary>.s

#after you insert your CFI logic into the assembly <test_binary>.s
as <test_binary>.s -o <test_binary>.out
ld <test_binary>.out -e _start -o <test_binary_with_cfi>
```

Task 1: protecting return (60%)

Implement a simple CFI policy to protect indirect return instructions. A simple policy can be: a return can jump to any instruction after a call instruction.

To test the effectiveness, use the ex01 binary. It should block the provided exploit. Note that unsuccessful CFI implementation might still cause the program to crash when feeding the exploit.

Task 2: protecting indirect calls (40%)

Implement a simple CFI policy to protect indirect function calls (e.g., call *%eax). A simple policy can be: an indirect call can jump to any function entry point.

To test the effectiveness, use the bof3 binary. It should block the provided exploit. Note that unsuccessful CFI implementation might still cause the program to crash when feeding the exploit.

You need to include the following in your report:

1. Important code snippets and explanations of how you implement these two protections.
2. Use the two programs to show that a) the protected binary can process normal inputs correctly without crashing; and b) when you provide a malicious input that hijacks the control flow, the protected binary can prevent it.

Rubrics:

For each task: Functionality (40%), Explanation (40%), Evaluation (20%)