Lecture 13
Control Flow Integrity Violation

Joshua Reynolds
NMSU RE
Spring 2024

Control Flow

How each thread of a process moves through the code, and decides what branches to take for loops, ifs, switches, and gotos.

Control Flow Integrity

Adversaries developing exploits seek to violate the property of **Control Flow Integrity**

Control Flow Integrity:

The control flow the program actually follows stays within the designed bounds and planned functionality.

Control Flow Integrity Violation

Attackers try to take control of processes by hijacking their control flow, and replacing it with their own:

- Creating a new thread running attacker code
- Misdirecting an existing thread into running attacker code
- Injecting extra commands via unsanitized user input (log4j)

Classic Buffer Overflow Attack

Remember stack frames save a return address. If an attacker can change this address, the program will jump somewhere else.

The stack grows from higher to lower addresses.

Buffers are written from low addresses to high addresses.

If you write too much data to a buffer, eventually it will overwrite the return address.

Buffer overflow attacks write shellcode into a buffer, then change the return address to point to their buffer!

Control Flow Integrity Protections - Stack Canaries

We’ve seen some of these in some of our decompiled examples.

On the stack right above the return address, a “canary” is added.

This canary may be a random number.

Then, before any ret instruction is allowed, that canary is checked.

How does this prevent the classic buffer overflow attack from the previous slide?

Getting around Canaries - Format Strings & Use-after-free

Format String Attacks:

You know %s, %d, ect. but what about %n?

What can we do with that?

How does it get around stack canaries?

Use-after-Free:

When you free memory, you should zero out the pointer to it. If you don’t, you may use it again. Attackers try to arrange things so that when you do use it again, you give them control. How?

Control Flow Integrity Protections - DEP

Data Execution Prevention, also called Write XOR Execute (W^X) memory protections are enforced by the processor.

Jumping to an address on the heap or the stack normally causes a segmentation fault.

Trying to use a function pointer to rewrite any code in the .text section will also cause a segmentation fault.

How does this protect control flow integrity?

Control Flow Integrity Defense - ASLR

Address Space Layout Randomization

When the OS creates a virtual address space, it places the stack, heap, and program at different addresses randomly chosen each time.

Makes exploit creation harder, because the address of return values and other useful pointers are not reliably the same every time.

Requires the programs to be compiled for **Position Independent Execution,** which is the default for most compilers now.

Defeating ASLR

To defeat ASLR, attackers need to leak a memory address from the process and then work backwards from there to find where every other address must be in relation to that address.

Defeating DEP and Stack Canaries - ROP

If the attacker can beat ASLR, there is an attack that can bypass DEP and stack canaries.

Return Oriented Programming (ROP):

Overwrite the return address with an array of valid addresses in the .text section

This address will be chosen to be a “**gadget**”

A gadget performs a few operations, then does a “ret” without a “leave”, grabbing the next valid return address from the injected array

A chain of gadgets, a **ROP Chain**, can be built to perform any task.

The gadgets in <stdlib.h> are **Turing Complete**

Developing a Control Flow Integrity Atack for your Assignment

You will build an attack using the `pwntools` python library to take over the control flow of a victim program.

In this case, the victim program simulates the type of assignment you probably wrote in a beginning CS course like 172. This version is vulnerable to an attack.

We'll do an activity to practice using pwntools.

You can find the code in the instructions