ENEE 457: Computer Systems Security

Buffer Overflow Countermeasures and RT-to-libc

Charalampos (Babis) Papamanthou

Department of Electrical and Computer Engineering
University of Maryland, College Park

Good programming practices

- Make sure you do bound checking
- Vulnerable C functions with strings:
    - strcpy
    - strcat
    - strcmp
    - gets

Address Space Layout Randomization

- Remember the reason we were able to guess the return address is that we assumed that the stack starts from the same address between two different executions of the program
- Or…the value of %ebp does not change between two executions of the program
- To get rid of this "bug", we can just tell the OS to randomize the address from where the stack begins every two executions of the program
  - Therefore, even if during debugging I get to see what the value of %ebp is, I will not know what the next value will be so that I can use it in my attack
- This countermeasure is called ASLR (address space layout randomization)
- For a 32-bit machine (2^32 addressable words), you can bruteforce and you can hit it.
- A system-wide option

StackGuard countermeasure

- Idea: Given source code, modify it so that it does not allow the buffer overflow attack to take place
- You need to store the initial return address to the heap, so that when you return you check to see whether the address you are returning to is the address that you initially stored
- You must store at the heap so that you cannot overwrite it.
- The observation is that you need to overwrite continuous portions of memory
- Put a random value before the return and check it before you return

Example code (secret should not be on the stack)

Buffer Overflow Attack Lecture (Part 3)

Computer & Network Security > Buffer Overflow

Stackguard Exercises

```
void foo (char *str)
{
    int guard;
    guard = secret;

    char buffer[12];
    strcpy (buffer, str);

    if (guard == secret)
        return;
    else
        error
}
```

int secret; → Heap
initialize with a random #

Stackguard

RA

ebp

random #

buffer

41:22 / 51:20

How about if we allocate secret on the heap?

Buffer Overflow Attack Lecture (Part 3)

Computer & Network Security > Buffer Overflow

Question

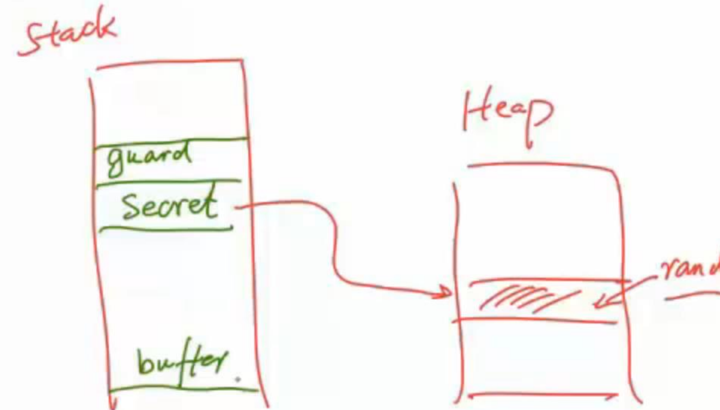
Question 1: (5 points). A programmer declares that the following code can defeat the buffer-overflow attack. Do you agree or not? Please give your justification. The secret only has 32 bit, which is quite weak as a secret, but we will ignore this issue in this question.

```
void func (char *str)
{
        int guard;
        int *secret = malloc (sizeof(int));
        *secret = generateRandomNumber();
        guard = *secret;

        char buffer[12];
        strcpy (buffer, str);

        if (guard != *secret) exit;

        return;
}
```



48:11 / 51:20

Instruction Set Randomization

- Instruction Set Randomization (ISR)
  - Each program has a *different* and *secret* instruction set
  - Use translator to randomize instructions at load-time
  - Attacker cannot execute its own code.

Non-executable stack

- Turn stack into non-executable
- OS needs to do it
- So even if the attacker puts shellcode into the stack, it will never run
- In particular, there is a bit in hardware that decides whether a block of memory can be executable or not
  - Turn off this bit for the stack
- Per application option
  - You can indicate that when you compile

Can I defeat non-executable stack?

- Try to use other people's code that is already in the system!
- Try to use shared library code already loaded in memory!
  - E.g., there is a `systemc("/bin/sh")` function somewhere
- Finding the address of system call is easy
- Passing the right argument is hard
- The problem is when systemc is called, a new stack frame will be allocated, and the argument should be stored at ebp+8
- So you need to overwrite ebp + 8 with the address of your string
- But how can you know the new value of ebp?
- And how can you overwrite it?

Return to-libc attack

- Instead of returning to the previous functions, return to a function that is shared and place the arguments appropriately