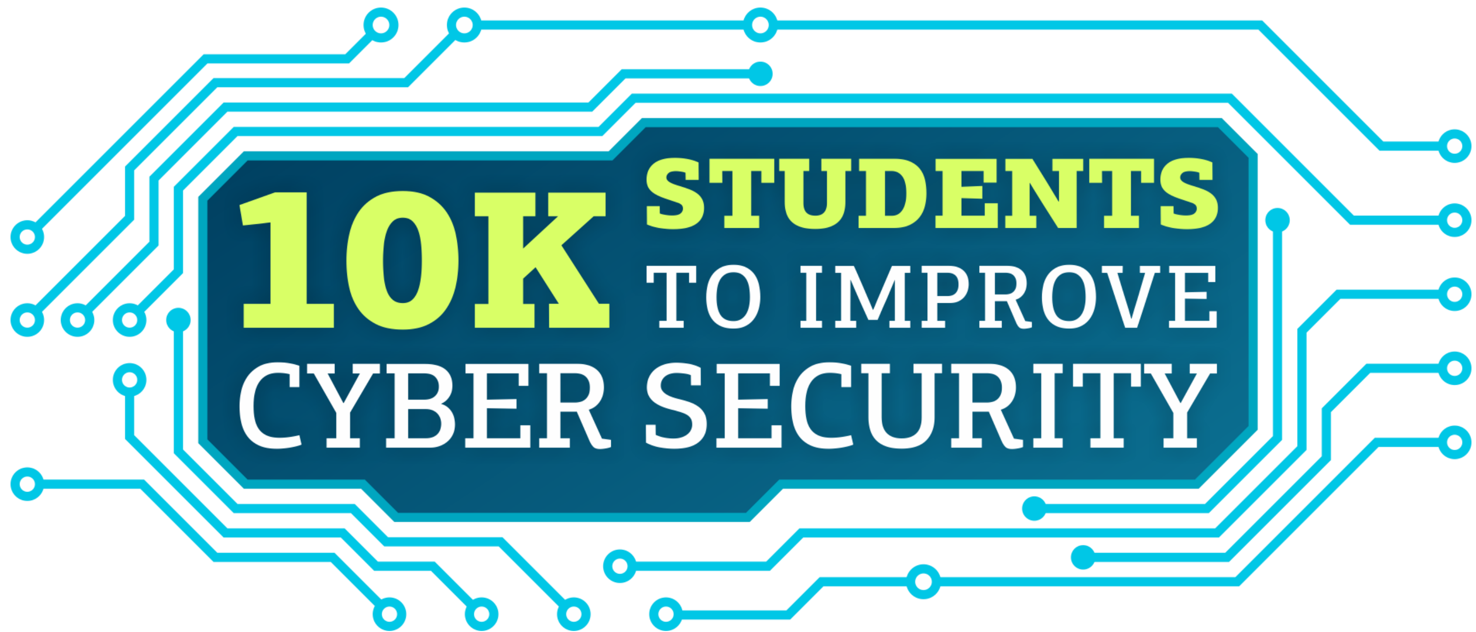

10K STUDENTS
TO IMPROVE
CYBER SECURITY

syssec

Part III
Counter measures

HOW DO WE STOP THE ATTACKS?

- The best defense is proper bounds checking
- but there are many C/C++ programmers and some are bound to forget

➔ Are there any *system* defenses that can help?

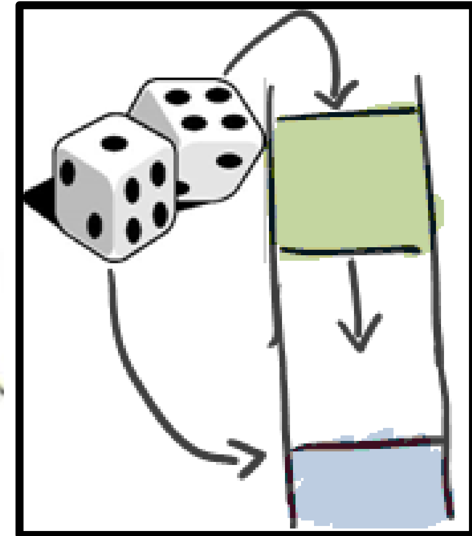
HOW DO WE STOP THE ATTACKS?

- A variety of tricks in combination

NX bit

Canaries

ASLR

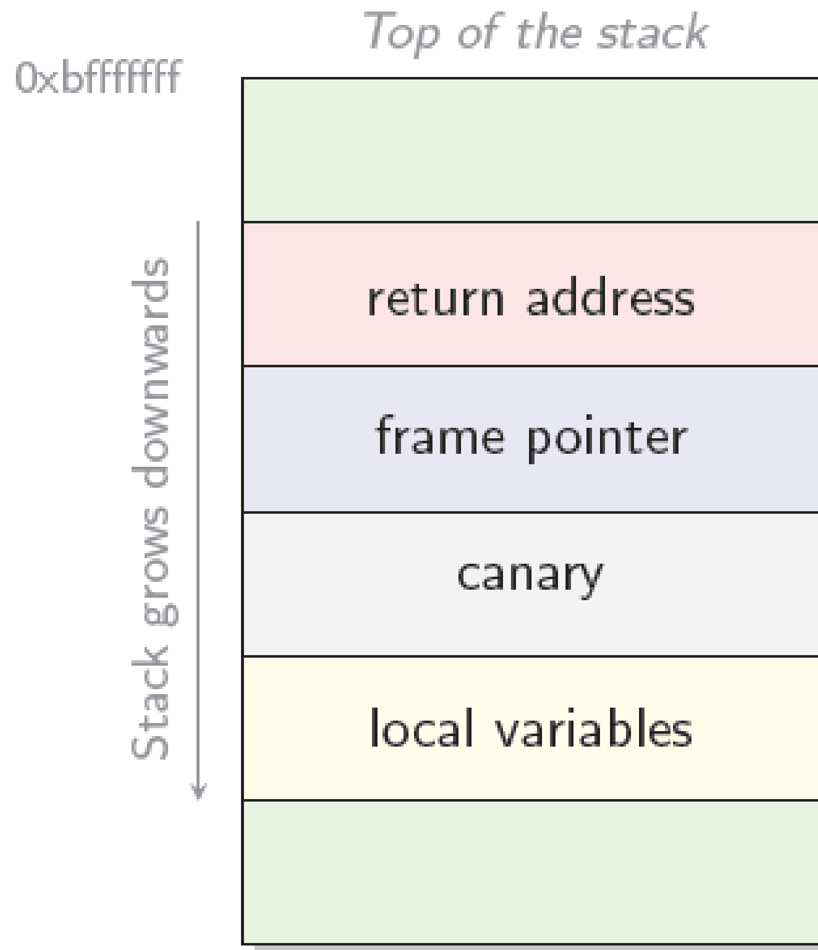


III.A
Canaries

Compiler-level techniques
Canaries

- Goal: make sure we detect overflow of return address
  - The functions' prologues insert a *canary* on the stack
  - The canary is a 32-bit value inserted between the return address and local variables
- Types of canaries:
  1. Terminator
  2. Random
  3. Random XOR
- The epilogue checks if the canary has been altered
- Drawback: requires recompilation

Canaries



How good are they?

- Assume random canaries protect the stack

VU UNIVERSITY AMSTERDAM

Can you still exploit this?

```
char gWelcome [] = "Welcome to our system! ";

void echo (int fd)
{
  int len;
  char name [64], reply [128];

  len = strlen (gWelcome);
  memcpy (reply, gWelcome, len);

  write_to_socket (fd, "Type your name: ");
  read (fd, name, 128);



  memcpy (reply+len, name, 64);

  write (fd, reply, len + 64);
  return;
}

void server (int socketfd) {
  while (1)
    echo (socketfd);
}
```

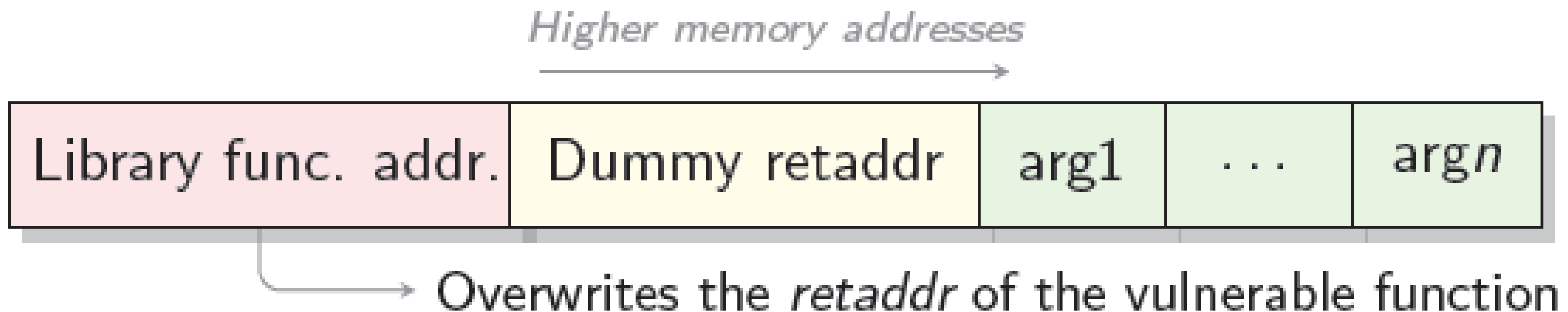
III.B
"DEP"

DEP / NX bit / W⊕X

- Idea: separate executable memory locations from writable ones
  - A memory page cannot be both writable and executable at the same time
- “Data Execution Prevention (DEP)”

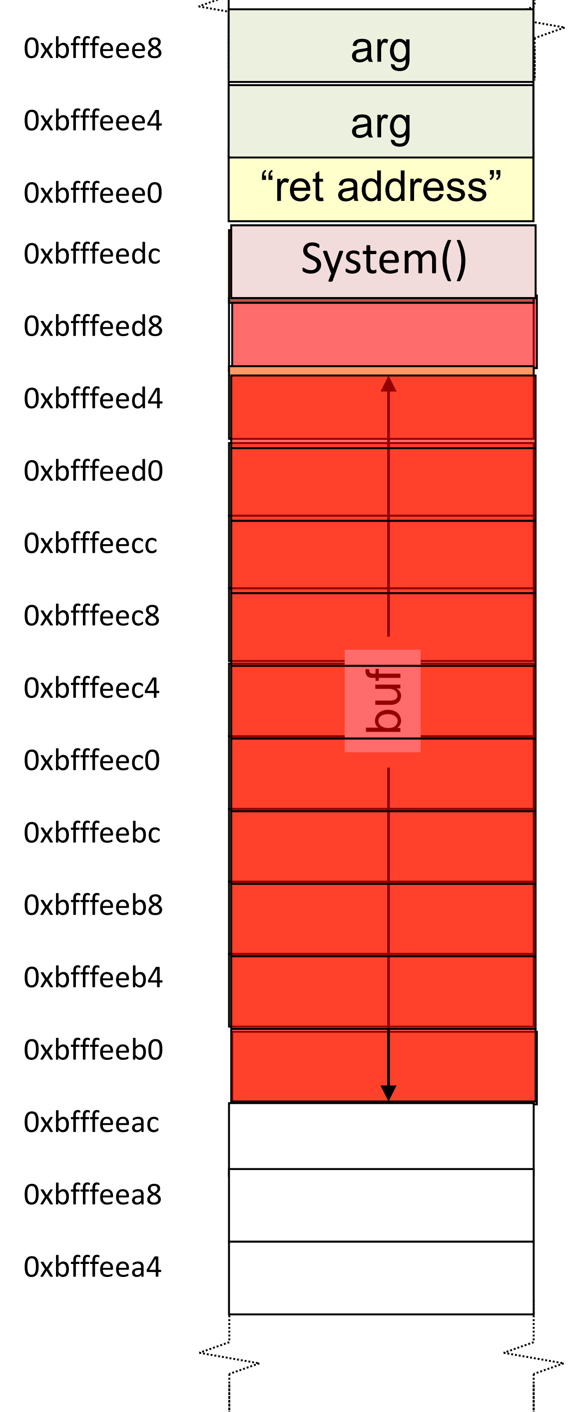
Bypassing W⊕X

- Return into libc
- Three assumptions:
  - We can manipulate a code pointer
  - The stack is writable
  - We know the address of a “suitable" library function (e.g., system())



Stack

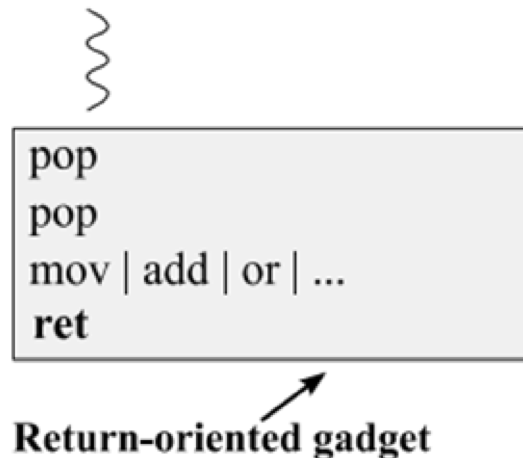
- Why the "ret address"?
- What could we do with it?



VU UNIVERSITY AMSTERDAM

Return Oriented Programming

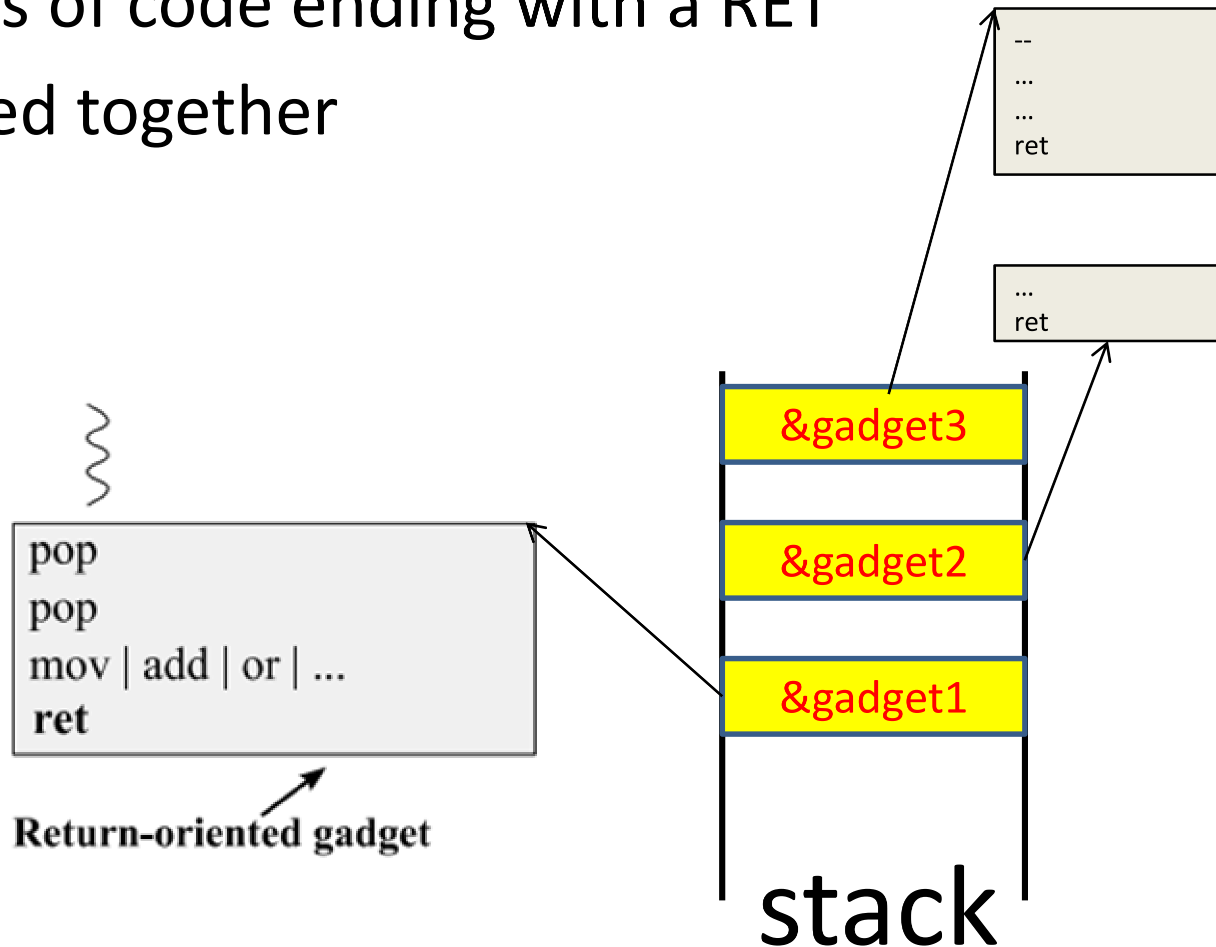
- ROP chains:
  - Small snippets of code ending with a RET
  - Can be chained together



VU UNIVERSITY AMSTERDAM

Return Oriented Programming

- ROP chains
  - Small snippets of code ending with a RET
  - Can be chained together



How good are they?

- Assume random canaries protect the stack
- Assume DEP prevents execution of the stack

Can you still exploit this?

```
char gWelcome [] = "Welcome to our system! ";

void echo (int fd)
{
  int len;
  char name [64], reply [128];

  len = strlen (gWelcome);
  memcpy (reply, gWelcome, len);

  write_to_socket (fd, "Type your name: ");
  read (fd, name, 128);



  memcpy (reply+len, name, 64);

  write (fd, reply, len + 64);
  return;
}

void server (int socketfd) {
  while (1)
    echo (socketfd);
}
```

III.C
ASLR

Let us make it a little harder still…

Address Space Layout Randomisation

- Idea:
  - Re-arrange the position of key data areas randomly (stack, .data, .text, shared libraries, . . .)
  - Buffer overflow: the attacker does not know the address of the shellcode
  - Return-into-libc: the attacker can't predict the address of the library function
  - Implementations: Linux kernel > 2.6.11, Windows Vista, . . .

ASLR: Problems

- 32-bit implementations use few randomisation bits
- An attacker can still exploit non-randomised areas, or rely on other information leaks (e.g., format bug)

- So… (I bet you saw this one coming)….

How good are they?

- Assume random canaries protect the stack
- Assume DEP prevents execution of the stack
- Assume ASLR randomized the stack *and* the start address of the code
  – but let us assume that all functions are still at the same relative offset from start address of code
  – (in other words: need only a single code pointer)

Can you still exploit this?

```
char gWelcome [] = "Welcome to our system! ";

void echo (int fd)
{
  int len;
  char name [64], reply [128];

  len = strlen (gWelcome);
  memcpy (reply, gWelcome, len);

  write_to_socket (fd, "Type your name: ");
  read (fd, name, 128);



  memcpy (reply+len, name, 64);

  write (fd, reply, len + 64);
  return;
}

void server (int socketfd) {
  while (1)
    echo (socketfd);
}
```

Finally

We constructed “weird machines”

- New spin on fundamental questions:
  - ➔ “What is computable?”
- Shellcode, ROP, Ret2Libc
  - ➔ Turing Complete

VU UNIVERSITY AMSTERDAM

That is all folks!

- We have covered quite a lot:
  - Simple buffer overflows
  - Counter measures
  - Counter counter measures
- Research suggests that buffer overflows will be with us for quite some time
- Best avoid them in your code!

VU UNIVERSITY AMSTERDAM