Oregon State University

COLLEGE OF ENGINEERING | School of Electrical Engineering and Computer Science

Buffer Overflow: An Overview

Buffer Overflow

Oregon State University
College of Engineering

- *"A condition at an interface under which **more input** can be placed into a buffer or data holding area **than the capacity allocated**, **overwriting** other information."*

- Used for exploitation
  - Inducing crashes
  - Taking control of program

Example

Oregon State University College of Engineering

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
  int valid = 0;
  char str1[8] = "ASECRET";
  char str2[8];

  gets(str2);

  if (strncmp(str1, str2, 8)==0)
     valid = 1;

  printf("VALID=%d", valid);

  return 0;
}
```

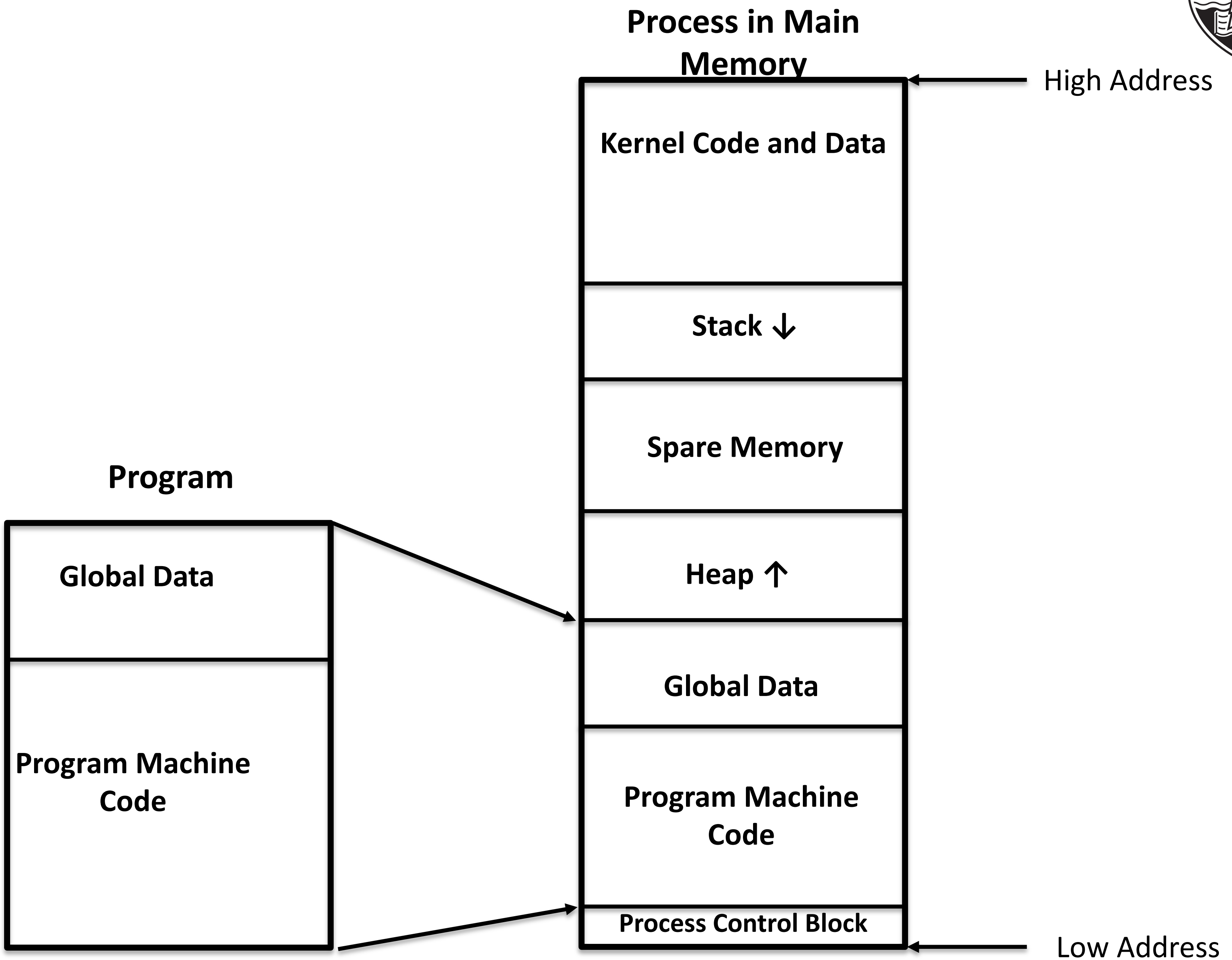
```
$ ./a.out
ASECRET
VALID=1

$ ./a.out
HELLO
VALID=0

$ ./a.out
OVERFLOWOVERFLOW
VALID=1
```

Why?

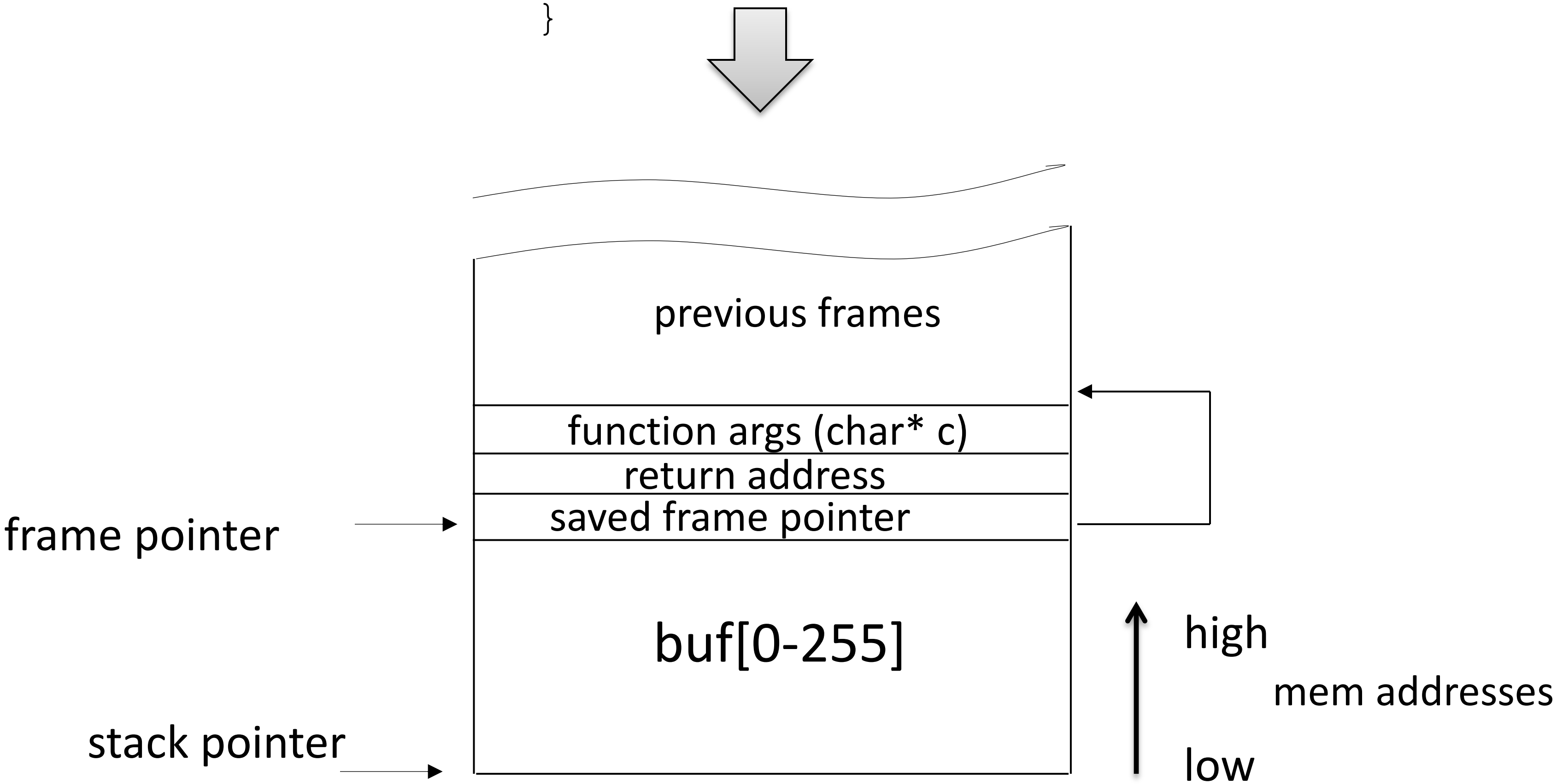
Process Memory Structure Review



Stack Structure Review

Oregon State University
College of Engineering

```
void foo(char* c)
{
    char buf[256];
    strcpy(buf, c);
}
```



Return to “OVERFLOW” Example

Oregon State University
College of Engineering

Stack	
	argv
	argc
	return address
	old base pointer
0x0000000	valid
RET	str1[4-7]
ASEC	str1[0-3]
. . .	str2[4-7]
. . .	str2[0-3]

Before gets(str2)

Stack	
	argv
	argc
	return address
	old base pointer
0x1000000	valid
FLOW	str1[4-7]
OVER	str1[0-3]
FLOW	str2[4-7]
OVER	str2[0-3]

After gets(str2)
str2=“OVERFLOWOVERFLOW
”

Stack Buffer Overflow

- Also called *Stack smashing*
  - Overflow when targeted buffer is located on **stack**, as a local variable in **stack frame** of a function
  - Overwrite return address/frame pointer to address of attack code in memory

- Example in next slide
  – Overwrite saved return address (RA)
  – E.g., overwrite saved RA with that of same function to re-execute it.

Example

Oregon State University
College of Engineering

```
#include <stdio.h>
#include <string.h>


void prompt(char * tag) {
    char inp[16];
    printf(“Enter value for %s: “, tag);
    gets(inp);
    printf(“Hello your %s is %s\n”, tag, inp);
}
int main(int argc, char *argv[]) {
    prompt(“name”);
}
```

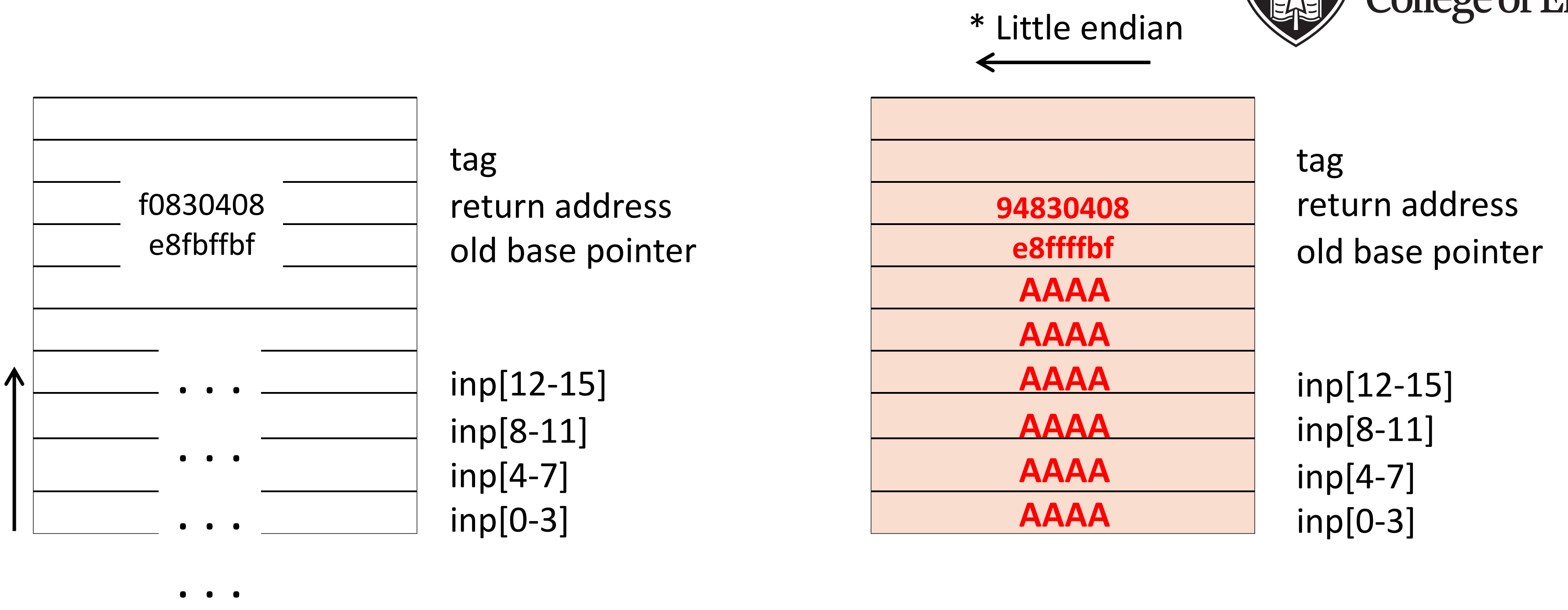
Example

- Run debugger to see prompt() is located at `0x08048394`
- *inp* is located 24 bytes under current frame ptr
  - So attacker will pad the string by this amount (e.g. 24 `A`'s)
- Choose a nearby stack location (e.g., `0xbfffbe8`) to overwrite current frame register
- Overwrite return address with `0x08048394`

- Combine this data together into binary string:

```
perl -e 'print pack("H*", hex string);'
```

Example



Before gets(..) call

After gets(..) call

```
$ perl -e 'print pack("H*", hex string);' | ./a.out
Enter value for name:
Hello your Re?pyy]uEA is AAAAAAAAAAAAAAAAAAAAAAuyu
Enter value for Kyyu:
Hello your Kyyu is NNNN
Segmentation fault
```

`prompt(..)` is called twice!

Stack Buffer Overflow

Oregon State University College of Engineering

- What if we want to do something more interesting than calling `prompt(..)` twice?

- Can set the return address to point to custom code held within the stack frame (*shellcode*).

Summary

Oregon State University
College of Engineering

- Buffer overflow is a serious threat to systems.

- It is possible to disrupt system and perform unauthorized actions using stack buffer overflow attacks.

Buffer Overflow: Defenses

Buffer Overflow Defenses

Oregon State University
College of Engineering

- 3-Steps to Buffer Overflow Exploits
  - A: Inject code via overflow
  - B: Change flow of control to injected code
  - C: Execute injected code

Defense Categories

- Compile-Time
  - Programming Language Choice (Step A)
  - Safe Coding (Step A)
  - Extensions / Safe Libraries (Step A)
  - Stack Protection (Step B)

- Run-Time
  - Executable Address Space Protection (Step C)
  - Address Space Randomization (Step B/C)
  - Guard Pages (Step B/C)

Compile-Time: Prog. Language Choice

- High-level (e.g. Java)
  - Strongly typed variables
  - Only permitted Operations
  - Range checking

- Downside, resource use

Compile-Time: Safe Coding

- Check for available space
- Code for graceful failure

Compile-Time: Safe Libs./Extensions

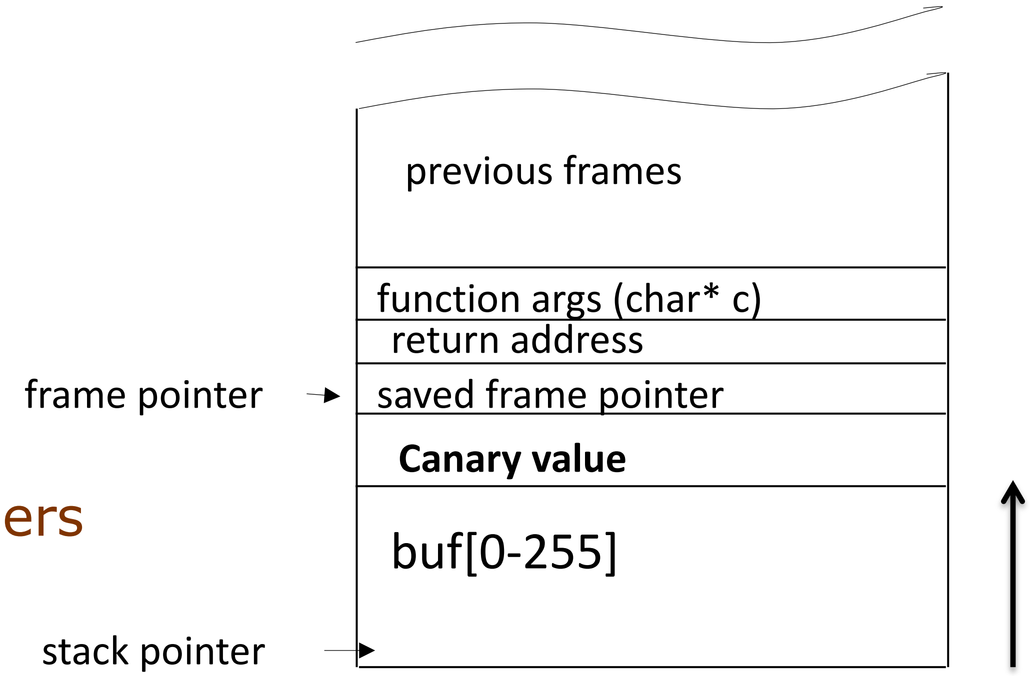
- Replace standard library routines (e.g. C strings) with safer versions
  - Rewrite source with new calls (like `strlcpy(..)`)

- Replace whole library – example "libsafe"
  - provides protection without recompilation
  - Puts additional checks to stop some buffer overflow attacks

Compile-Time: Stack Protection

Oregon State University
College of Engineering

- Check stack frame corruption
- StackGuard
  - Canary value
  - Included in newer gcc
  - No source code change
- StackGuard Cons
  - Needs recompilation
  - Changed stack structure -> change to debuggers
- Canary Requirements?
  - Different for each system
  - Unpredictable



Compile-Time: Stack Protection

- Return Address Defender (RAD) and StackShield
  - Copy return address (RA)
  - Safe location – called RAR by RAD folks
  - GCC option

- Con: Needs recompilation
- Pro:
  - No source code change
  - No changes to stack structure

Run-Time: Executable Address Space Protection

- Block execution of code on stack and heap
  - Prevents Step C
- `No-execute` bit
- Assume executable code held elsewhere
- Standard in newer O.S.'s (Vista+, Linux) and 64-bit processors
- Drawback: some legitimate uses for executing code on stack

Run-Time: Address Space Randomization

- Stack buffer overflow: need to predict address of buffer in memory (decide what is proper RA value)
- Change address stack location random per process
- Address range is large (32 bit), provides much variation. Larger than most vulnerable buffers.
- Therefore NOP-sled will not work (cannot get it large enough to handle large range.)
- Prevents steps B/C of buffer overflow attack

Run-Time

Oregon State University
College of Engineering

- Guard Pages
  - Introduce additional pages between critical regions of memory
    - Mark them as illegal
  - Can thwart buffer overflow exploits in global data

Return to System Call Attack

Oregon State University College of Engineering

- Non-executable stack defense
  - Cannot execute code held in stack

- Make code call library function through a set of memory location manipulations

- RA changed to jump to existing system library function, e.g. `system("shell command line")` in order to launch shell commands.
  - Attack contained in parameters, e.g. "command line"

- Defend by randomizing stack and system libraries

Summary

Oregon State University
College of Engineering

- Buffer overflow is a serious threat to systems.
- It is possible to disrupt system and perform unauthorized actions using stack buffer overflow attacks.
- Shellcode can be used to modify execution of a vulnerable program.
- There exist compile- and run-time protections against these attacks.
- Buffer Overflows can happen on Heap and Global Data sections as well