CS326 – Systems Security

Lecture 10
Introduction to Software Security

Elias Athanasopoulos
eliasathan@cs.ucy.ac.cy
Recall
Basic Problem

Oscar can see the message (confidentiality)
Oscar can modify the message (integrity)
Recall
Cryptographic Approach

m: plaintext
c: ciphertext
k: key

Cryptosystem
Encryption algorithm
Decryption algorithm
Key(s) involvement

Alice $\rightarrow$ Encrypt $\rightarrow$ Internet $\rightarrow$ Decrypt $\rightarrow$ Bob

Secure Channel

K
Implicit Assumption
Basic Problem

Code Runs Correctly?

Alice

Encrypt

c

Internet

Oscar
c

Code Runs Correctly?

Decrypt

m

Bob

Secure Channel

K

K
Software Roadmap

- Software properties
- Software bugs
- Attacks and exploitation
- Defenses
Software

• Programs are written in a high-level language
  – C, C++, Java, C#, Ruby, Python
• They are compiled for execution
  – Machine code (unmanaged code)
  – Virtual-machine code, such as JVM, (managed code)
• Different architectures exhibit different properties in executing software
  – Some generic concepts apply to all
Software Composition

• Data
  – Static pre-defined and allocated at compile-time
  – Dynamically allocated and short-lived (usually at the stack of the program)
  – Dynamically allocated and long-lived (usually at the heap of the program)

• Code
  – Known at compile-time (ahead of time)
  – Generated at run-time (just-in time) using a JIT engine
Compilation, Loading and Execution

• The compiler prepares the program
  – Creates a binary file stored in the disk
  – Binary contains code and data
  – Binary has different sections
  – Binary may have additional dependencies (shared libraries)

• The loader reads the binary and maps it to memory
  – Resolves symbols
  – Loads shared libraries

• The OS creates a structure called *process*
  – A process is a virtual concept of an executing program
  – A binary (stored in the disk) has a different layout from an executing process (mapped in memory)
Processes

• Programs are executed with the help of the OS
• The OS creates an internal structure (called virtual process) for running the program
• Each virtual process has several things
  – Resources (opened files and sockets, mapped files, etc.)
  – State (page tables, register values, memory contents, etc.)
  – A pointer to the next instruction (program counter)
• Processes execute as they are alone
  – All (virtual) memory is available for a running process
  – The OS can execute several processes, concurrently
Virtual Memory

• A process works using memory
• Memory is given by the OS in a number of virtual pages (a few kilobytes each)
• Physical memory is mapped to virtual pages
  – Every process has its own map (page tables)
  – If physical memory runs out, it is swapped to disk
• Not all pages are loaded at once
  – Code pages are loaded when needed (using pagefaults)
  – Additional pages for data are allocated upon request
• Accessing a memory address that is not contained in a loaded page causes a pagefault
  – Unless the OS can load the page (i.e., a code page) or handle it somehow, the process will crash
Virtual Memory

• It is practical to have different memory areas
  – Code region has pages having code
  – Data region has pages having data
  – Stack has pages hosting temporary data
  – Heap has pages with dynamically allocated data

• Pages have permissions
  – Some enforced by the hardware (MMU)
  – NX-bit
Example of a C program

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    int ary[5] = {1, 2, 3, 4, 5};
    fprintf(stderr, "The fifth number of ary is: %d\n", ary[5]);
    return 1;
}
```
Out of bounds access!

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    int ary[5] = {1, 2, 3, 4, 5};
    fprintf(stderr, "The fifth number of ary is: %d\n", ary[5]);
    return 1;
}
```
Machine code (32-bit)

0804843b <main>:

0804843b:       8d 4c 24 04       lea  0x4(%esp),%ecx
0804843f:       83 e4 f0          and  $0xfffffffff0,%esp
08048442:       ff 71 fc          pushl -0x4(%ecx)
08048445:       55               push  %ebp
08048446:       89 e5            movl  %esp,%ebp
08048448:       51               push  %ecx
08048449:       83 ec 24          sub  $0x24,%esp
0804844c:       c7 45 e4 01 00 00 00  movl  $0x1,-0x1c(%ebp)
08048453:       c7 45 e8 02 00 00 00  movl  $0x2,-0x18(%ebp)
0804845a:       c7 45 ec 03 00 00 00  movl  $0x3,-0x14(%ebp)
08048461:       c7 45 f0 04 00 00 00  movl  $0x4,-0x10(%ebp)
08048468:       c7 45 f4 05 00 00 00  movl  $0x5,-0x0c(%ebp)
0804846f:       8b 55 f8          mov  -0x8(%ebp),%edx
08048472:       a1 20 a0 04 08      mov  0x804a020,%eax
08048477:       83 ec 04          sub  $0x4,%esp
0804847a:       52               push  %edx
0804847b:       68 20 85 04 08      push  $0x8048520
08048480:       50               push  %eax
08048481:       e8 9a fe ff ff      call  8048320 <fprintf@plt>
08048486:       83 c4 10          add  $0x10,%esp
08048489:       b8 01 00 00 00      mov  $0x1,%eax
0804848e:       8b 4d fc           mov  -0x4(%ebp),%ecx
08048491:       c9               leave
08048492:       8d 61 fc           lea  -0x4(%ecx),%esp
08048495:       c3               ret
Machine Code (64-bit)

0000000000400546 <main>:

400546:  55                      push %rbp
400547:  48 89 e5               mov %rsp,%rbp
40054a:  48 83 ec 30            sub $0x30,%rsp
40054d:  89 7d dc              mov %edi,-0x24(%rbp)
400550:  48 89 75 d0            mov %rsi,-0x30(%rbp)
400553:  c7 45 e0 01 00 00 00   movl $0x1,-0x20(%rbp)
40055b:  c7 45 e4 02 00 00 00   movl $0x2,-0x1c(%rbp)
400561:  c7 45 e8 03 00 00 00   movl $0x3,-0x18(%rbp)
400569:  c7 45 ec 04 00 00 00   movl $0x4,-0x14(%rbp)
40056f:  c7 45 f0 05 00 00 00   movl $0x5,-0x10(%rbp)
400577:  8b 55 f4              mov -0xc(%rbp),%edx
40057d:  48 8b 05 be 0a 20 00   mov 0x200abe(%rip),%rax #<stderr@_GLIBC_2.2.5>
400583:  be 28 06 40 00          mov $0x400628,%esi
40058a:  48 89 c7              mov %rax,%rdi
40058d:  b8 00 00 00 00 00     mov $0x0,%eax
400591:  e8 ac fe ff ff         callq 400440 <fprintf@plt>
400596:  b8 01 00 00 00 00     mov $0x1,%eax
40059c:  c9                     leaveq
40059d:  c3                     retq
Machine Code
(32-bit, hardened)

00000600 <main>:
600: 8d 4c 24 04      lea 0x4(%esp),%ecx
604: 83 e4 f0       and $0xffffffff,%esp
607: ff 71 fc       pushl -0x4(%ecx)
60a: 55             push %ebp
60b: 89 e5          mov %esp,%ebp
60d: 53             push %ebx
60e: 51             push %ecx
60f: 83 ec 30       sub $0x30,%esp
612: e8 7d 00 00 00  call 694 <__x86.get_pc_thunk.ax>
617: 05 b5 19 00 00  add $0x19b5,%eax
61c: 89 ca          mov %ecx,%edx
61e: 8b 52 04       mov 0x4(%edx),%edx
621: 89 55 d4       mov %edx,-0x2c(%ebp)
624: 65 8b 1d 14 00 00 00  mov %gs:0x14,%ebx
62b: 89 5d f4       mov %ebx,-0xc(%ebp)
62e: 31 db          xor %ebx,%ebx
630: c7 45 e0 01 00 00 00  movl $0x1,-0x20(%ebp)
637: c7 45 e4 02 00 00 00  movl $0x2,-0x1c(%ebp)
63e: c7 45 e8 03 00 00 00  movl $0x3,-0x18(%ebp)
645: c7 45 ec 04 00 00 00  movl $0x4,-0x14(%ebp)
64c: c7 45 f0 05 00 00 00  movl $0x5,-0x10(%ebp)
653: 8b 4d f4       mov -0xc(%ebp),%ecx
656: 8b 90 10 00 00 00  mov 0x10(%eax),%edx
Machine Code
(32-bit, hardened)

```
65c:  8b 12   mov (%edx),%edx
65e:  83 ec 04  sub $0x4,%esp
661:  51         push %ecx
662:  8d 88 74 e7 ff ff  lea -0x188c(%eax),%ecx
668:  51         push %ecx
669:  52         push %edx
66a:  89 c3  mov %eax,%ebx
66c:  e8 0f fe ff ff  call 480 <.plt.got+0x20>
671:  83 c4 10  add $0x10,%esp
674:  b8 01 00 00 00  mov $0x1,%eax
679:  8b 5d f4  mov -0xc(%ebp),%ebx
67c:  65 33 1d 14 00 00 00  xor %gs:0x14,%ebx
683:  74 05        je 68a <main+0x8a>
685:  e8 86 00 00 00 00  call 710 <__stack_chk_fail_local>
688:  8d 65 f8  lea -0x8(%ebp),%esp
68a:  59         pop %ecx
68d:  5b         pop %ebx
68f:  5d         pop %ebp
690:  8d 61 fc  lea -0x4(%ecx),%esp
693:  c3         ret
```
Vocabulary

• Vulnerability
  – A software error (also known as bug) that can potentially allow someone to take advantage of the vulnerable program

• Exploit
  – The process of controlling a program by taking advantage of one or more vulnerabilities
  – Not all vulnerabilities can be exploited
Vocabulary

• Arbitrary Code Execution
  – The state of an exploit where an attacker can execute a program of their choice

• Shellcode
  – A machine code that a vulnerable program executes and serves the purposes of the attacker
  – Spawn a shell (can be remote), download malware, create a hidden account, manipulate software, etc.
  – Heavily architecture dependent
High-level Idea

Vulnerable Web Server

Process Request

Exploit Bug(s)

Run Shellcode

Malicious Request
(contains code to trigger the bug and shellcode)

Remote Shell
(open socket back to the attacker’s host)

Attacker
High-level Idea

Vulnerable Web Browser

Process Response

Exploit

Run Shellcode

Malicious Web Site

Malicious Response
(contains code to trigger the bug and shellcode)

Remote Shell
(open socket back to the attacker’s host)
Software Exploitation

- The victim program has vulnerabilities
  - Can be a program executing in user-space
  - Can be the OS
- Bugs can be triggered using malicious inputs
  - Inputs can be sent over the network (remote attacker)
  - Inputs can be sent locally (local attacker)
- Triggering the bugs can lead to arbitrary code execution
- Arbitrary code execution can run the shellcode