University of BRISTOL

Computer System B- Security

Introduction to Memory-layout and Assembly

Sanjay Rawat

bristol.ac.uk

Lecture(s) Agenda

- Motivation (non-technical)
- Background
  - C to assembly
  - Memory layout

Motivation

- Bruce Schneier wrote

  "Security is a chain; it's only as secure as the weakest link."

Motivation

- Bruce Schneier wrote

  "Security is a chain; it's only as secure as the weakest link."

- cryptography is already strong (mathematically!)

Motivation

- Bruce Schneier wrote

  "Security is a chain; it's only as secure as the weakest link."

- cryptography is already strong (mathematically!)
- problem lies in software, hardware, networks etc.

Motivation

- Bruce Schneier wrote

  "Security is a chain; it's only as secure as the weakest link."

- cryptography is already strong (mathematically!)
- problem lies in software, hardware, networks etc.
- In this module, we'll focus on software parts (Network follows)

Motivation

- Bruce Schneier wrote

  "Security is a chain; it's only as secure as the weakest link."

- cryptography is already strong (mathematically!)
- problem lies in software, hardware, networks etc.
- In this module, we'll focus on software parts (Network follows)
- Why?

Software Security *w.r.t.* CIA

Software Security *w.r.t.* CIA

- What is a computer program?
  - A computer program is a collection of instructions that performs a specific task when executed by a computer. Most computer devices require programs to function properly. –Wikipedia

Software Security *w.r.t.* CIA

- What is a computer program?
  - A computer program is a collection of instructions that performs a specific task when executed by a computer. Most computer devices require programs to function properly. –Wikipedia
- For an application, (explicit) *security features* are well defined » application of Cryptography, keeping in mind its CIA properties.

Software Security *w.r.t.* CIA

- What is a computer program?
  - A computer program is a collection of instructions that performs a specific task when executed by a computer. Most computer devices require programs to function properly. –Wikipedia
- For an application, (explicit) *security features* are well defined » application of Cryptography, keeping in mind its CIA properties.
- *Intended behavior must match the implemented behavior*

Software Security *w.r.t.* CIA

- What is a computer program?
  - A computer program is a collection of instructions that performs a specific task when executed by a computer. Most computer devices require programs to function properly. –Wikipedia
- For an application, (explicit) *security features* are well defined » application of Cryptography, keeping in mind its CIA properties.
- *Intended behavior must match the implemented behavior*
- *Intrinsic properties* of the code (programming language) may interfere with these intended CIA!

Security Flaws

- Software Defects:
  - A software defect is the encoding of a human error into the software, including omissions.

Security Flaw:

- A *security flaw* is a software defect that poses a potential security risk.
- Eliminating software defects eliminate security flaws.

Software Vulnerability

Software Vulnerability

- WEAKNESS: a type of behavior that has the potential for allowing an attacker to violate the intended security policy, if the behavior is made accessible to the attacker. e.g. strcpy()

Software Vulnerability

- WEAKNESS: a type of behavior that has the potential for allowing an attacker to violate the intended security policy, if the behavior is made accessible to the attacker. e.g. strcpy()

- VULNERABILITY: a set of one or more related weaknesses within a specific software product or protocol that allows an actor to access resources or behaviors that are outside of that actor's control sphere, i.e., that do not provide appropriate protection mechanisms to enforce the control sphere.

Software Vulnerability

- WEAKNESS: a type of behavior that has the potential for allowing an attacker to violate the intended security policy, if the behavior is made accessible to the attacker. e.g. strcpy()

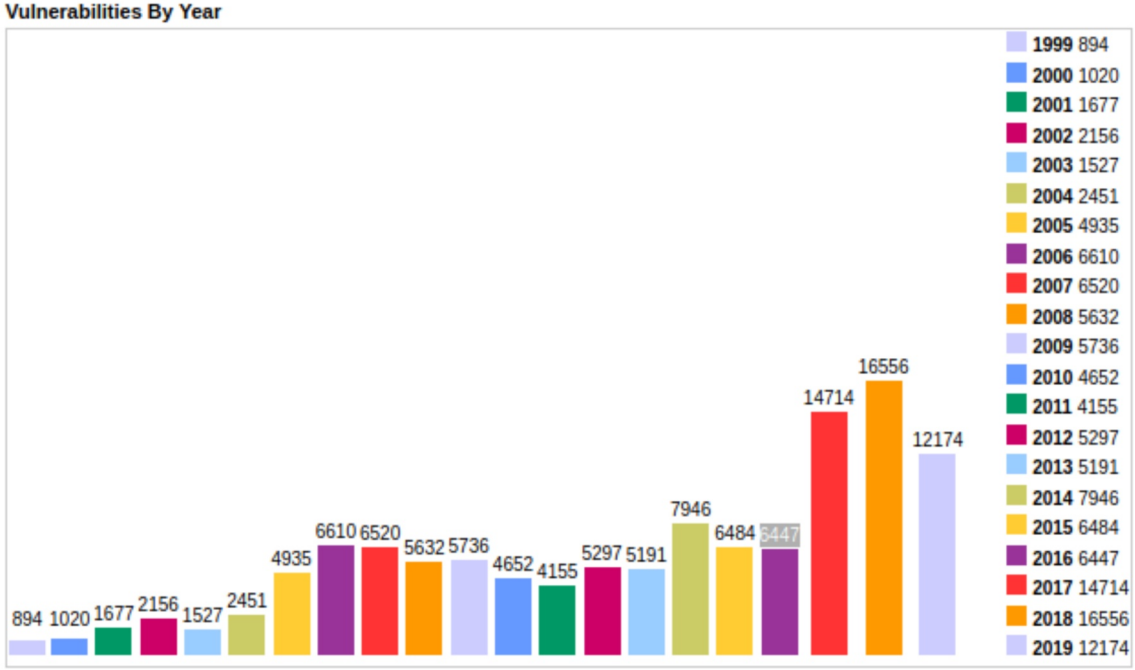
- VULNERABILITY: a set of one or more related weaknesses within a specific software product or protocol that allows an actor to access resources or behaviors that are outside of that actor's control sphere, i.e., that do not provide appropriate protection mechanisms to enforce the control sphere.

- **Vulnerability** is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, and attacker capability to exploit the flaw

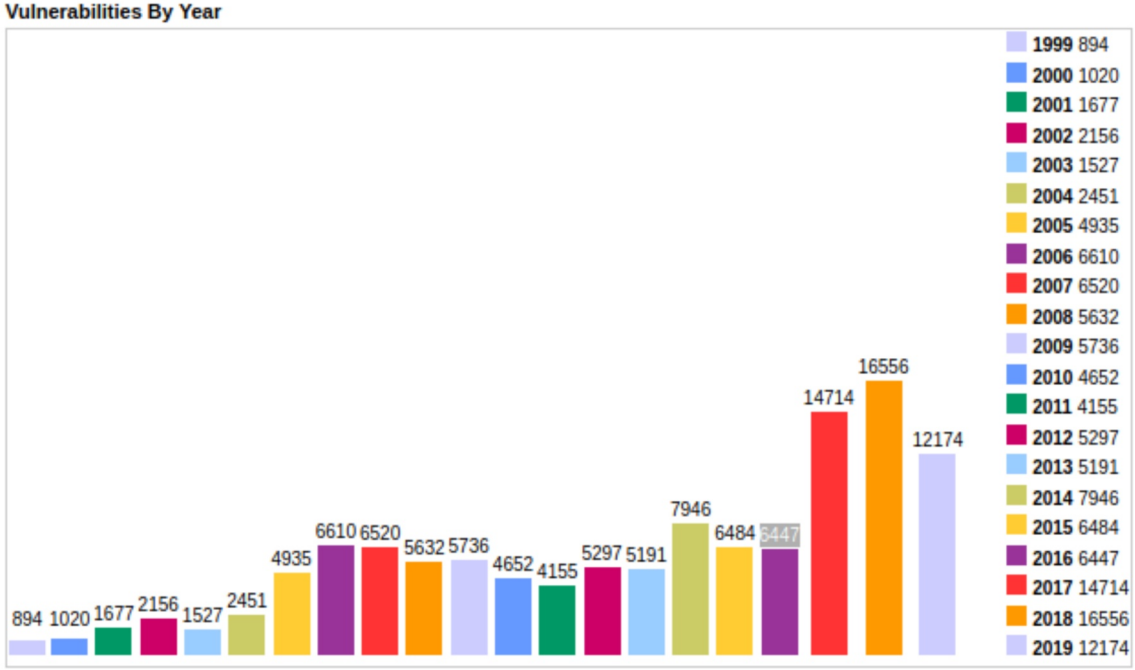
Exploits

- Exploit:
  - An *exploit* is a piece of software or technique that takes advantage of a security vulnerability to violate an explicit or implicit security policy.
  - Vulnerabilities in software are subject to exploitation.
  - Exploits can take many forms, including worms, viruses, and trojans.

Why do we care?



Why do we care?



Vulnerabilities by year (http://www.cvedetails.com)

Memory Corruption Vulnerabilities

- WYSINWYX: What You See Is Not What You eXecute by G. Balakrishnan et. al.
  - Higher level code -> low-level representation
  - Seemingly separate variables -> contiguous memory addresses
- Contiguous memory locations allow for boundary violations!
- We will talk about them in detail next week!

Side effects

Side effects

- Over/underflow

Side effects

- Over/underflow
- Sensitive data corruption

Side effects

- Over/underflow
- Sensitive data corruption
- Control data corruption (control hijacking)

Side effects

- Over/underflow
- Sensitive data corruption
- Control data corruption (control hijacking)

If done properly-exploit

Side effects

- Over/underflow
- Sensitive data corruption
- Control data corruption (control hijacking)

If done properly-exploit

Otherwise crash!

Generic Approach to Vulnerability Detection

1. Find the weakness in the Software (sink)
2. Find the source of *tainted input* (source)
3. Find the tainted path from source to sink.

Generic Approach to Vulnerability Detection

1. Find the weakness in the Software (sink)
2. Find the source of *tainted input* (source)
3. Find the tainted path from source to sink.

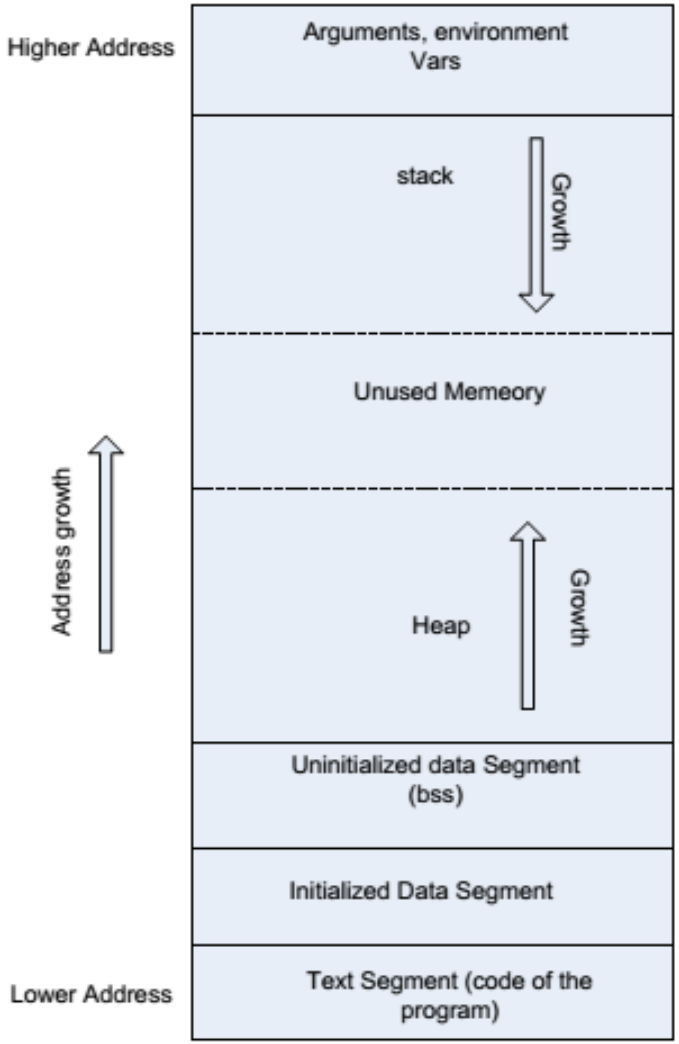
Mitigations

- Mitigation:
  - *Mitigations* are methods, techniques, processes, tools, or runtime libraries that can prevent or limit exploits against vulnerabilities.
  - At the source code level, a mitigation might be replacing an unbounded string copy operation with a bounded one.
  - At a system or network level, a mitigation might involve turning off a port or filtering traffic to prevent an attacker from accessing a vulnerability.

Understanding Memory Layout and Intel x86 ISA

Typical Memory Layout

- Windows PE and Linux ELF file formats
- Main segments of a compiled code:
  - The executable code (text segment)
  - The initialized data (data segment)
  - The uninitialized data (bss segment)
  - The heap (dynamic memory allocation)
  - The executable code and data of needed shared libraries (dynamically loaded into the space)
  - The program stack


Higher Address
Arguments, environment
Vars
stack
Growth
Unused Memeory
Address growth
Heap
Growth
Uninitialized data Segment
(bss)
Initialized Data Segment
Lower Address
Text Segment (code of the
program)

x86 Assembly (32-bit)

- 6 general purpose registers
  - EAX, EBX, ECX, EDX, ESI EDI
- 2 special registers ESP, EBP
- 1 very special register EIP
- 100s of instructions
  - Data movement
  - Arithmetic
  - jump

x86 Assembly (32-bit)

- 6 general purpose registers
  - EAX, EBX, ECX, EDX, ESI EDI
- 2 special registers ESP, EBP
- 1 very special register EIP
- 100s of instructions
  - Data movement
  - Arithmetic
  - jump

%rax	X86-64
%rcx	
%rdx	
%rbx	
%rsi	
%rdi	
%rsp	
%rbp	
%r8	
%r9	
%r10	
%r11	
%r12	
%r13	
%r14	
%r15	

Generic code syntax/semantic

- Two address mode
  - `Opcode operand1, operand2`
  - Intel- operand1 is destination, `mov eax, 10`
  - AT&T- operand2 is destination, `mov 10, eax`
- `Arithmetic`
  - `Add, sub, mul… add eax, ebx -> eax=eax+ebx`
- `Data movement`
  - `Mov, push pop… mov eax, 10 -> eax=10; mov eax, [ebx]`
- `Jump`
  - `Jmp <address>`

Calling conventions x86-32

- cdecl – parameters are pushes onto the stack, caller cleans the stack
- stdcall- parameters are pushes onto the stack, callee cleans the stack
- fastcall- first 3 parameters passed in EAX, EDX, ECX, rest on stack.
- thiscall- OOP, pointer to class object in ecx, rest are on stack.

Calling conventions x86-64

- Microsoft x64 calling convention
  - Registers RCX, RDX, R8, R9 for the first four integer or pointer arguments (in that order)
  - XMM0, XMM1, XMM2, XMM3 are used for floating point arguments
  - Additional arguments are pushed onto the stack (right to left)

- Unix like systems
  - The first six integer or pointer arguments are passed in registers RDI, RSI, RDX, RCX, R8, R9
  - XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6 and XMM7 are used for certain floating point arguments
  - Additional arguments are passed on the stack
  - The return value is stored in RAX and RDX

Calling conventions x86-64

- Microsoft x64 calling convention
  - Registers RCX, RDX, R8, R9 for the first four integer or pointer arguments (in that order)
  - XMM0, XMM1, XMM2, XMM3 are used for floating point arguments
  - Additional arguments are pushed onto the stack (right to left)

- Unix like systems
  - The first six integer or pointer arguments are passed in registers RDI, RSI, RDX, RCX, R8, R9
  - XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6 and XMM7 are used for certain floating point arguments
  - Additional arguments are passed on the stack
  - The return value is stored in RAX and RDX

If any of these argument registers are ***read first***, before writing, it is fastcall / x86-64 calling convention

C to assembly

```
void hello()
{
    printf("Hello from Hello!\n");
}
int buf_copy(char* string)
{
    char buffer[50];
    int dummy=200;
    strcpy(buffer, string);
    printf("%d - %s\n", dummy, buffer);
    return 0;
}
int main(int argc, char **argv)
{
    char name[]="World!";
    hello();
    printf("Hello %s\n",name);
    buf_copy(argv[1]);
    return 0;
}
```

```
int main(int argc, char
**argv)
{
    char name[]="World!";
    hello();
    printf("Hello %s\n",
name);
    buf_copy(argv[1]);
    return 0;
}
```

x86-64 binary

```
0000000000000734 <main>:
 734:   55                      push   %rbp
 735:   48 89 e5                mov    %rsp,%rbp
 738:   48 83 ec 20             sub    $0x20,%rsp
 73c:   89 7d ec                mov    %edi,-0x14(%rbp)
 73f:   48 89 75 e0             mov    %rsi,-0x20(%rbp)
 743:   c7 45 f9 57 6f 72 6c    movl   $0x6c726f57,-0x7(%rbp)
 74a:   66 c7 45 fd 64 21       movw   $0x2164,-0x3(%rbp)
 750:   c6 45 ff 00             movb   $0x0,-0x1(%rbp)
 754:   b8 00 00 00 00          mov    $0x0,%eax
 759:   e8 7c ff ff ff          callq  6da <hello>
 75e:   48 8d 45 f9             lea    -0x7(%rbp),%rax
 762:   48 89 c6                mov    %rax,%rsi
 765:   48 8d 3d c3 00 00 00    lea    0xc3(%rip),%rdi
 76c:   b8 00 00 00 00          mov    $0x0,%eax
 771:   e8 3a fe ff ff          callq  5b0 <printf@plt>
 776:   48 8b 45 e0             mov    -0x20(%rbp),%rax
 77a:   48 83 c0 08             add    $0x8,%rax
 77e:   48 8b 00                mov    (%rax),%rax
 781:   48 89 c7                mov    %rax,%rdi
 784:   e8 64 ff ff ff          callq  6ed <buf_copy>
 789:   b8 00 00 00 00          mov    $0x0,%eax
 78e:   c9                      leaveq
 78f:   c3                      retq
```

```
int main(int argc, char
**argv)
{
    char name[]="World!";
    hello();
    printf("Hello %s\n",
name);
    buf_copy(argv[1]);
    return 0;
}
```

x86-64 binary

```
0000000000000734 <main>:
 734:   55                      push   %rbp
 735:   48 89 e5                mov    %rsp,%rbp
 738:   48 83 ec 20             sub    $0x20,%rsp
 73c:   89 7d ec                mov    %edi,-0x14(%rbp)
 73f:   48 89 75 e0             mov    %rsi,-0x20(%rbp)
 743:   c7 45 f9 57 6f 72 6c    movl   $0x6c726f57,-0x7(%rbp)
 74a:   66 c7 45 fd 64 21       movw   $0x2164,-0x3(%rbp)
 750:   c6 45 ff 00             movb   $0x0,-0x1(%rbp)
 754:   b8 00 00 00 00          mov    $0x0,%eax
 759:   e8 7c ff ff ff          callq  6da <hello>
 75e:   48 8d 45 f9             lea    -0x7(%rbp),%rax
 762:   48 89 c6                mov    %rax,%rsi
 765:   48 8d 3d c3 00 00 00    lea    0xc3(%rip),%rdi
 76c:   b8 00 00 00 00          mov    $0x0,%eax
 771:   e8 3a fe ff ff          callq  5b0 <printf@plt>
 776:   48 8b 45 e0             mov    -0x20(%rbp),%rax
 77a:   48 83 c0 08             add    $0x8,%rax
 77e:   48 8b 00                mov    (%rax),%rax
 781:   48 89 c7                mov    %rax,%rdi
 784:   e8 64 ff ff ff          callq  6ed <buf_copy>
 789:   b8 00 00 00 00          mov    $0x0,%eax
 78e:   c9                      leaveq
 78f:   c3                      retq
```

leaveq: mov %rbp, %rsp
pop %rbp

prologue

```
int main(int argc, char
**argv)
{
    char name[]="World!";
    hello();
    printf("Hello %s\n",
name);
    buf_copy(argv[1]);
    return 0;
}
```

x86-64 binary

```
0000000000000734 <main>:
 734:   55                      push   %rbp
 735:   48 89 e5                mov    %rsp,%rbp
 738:   48 83 ec 20             sub    $0x20,%rsp
 73c:   89 7d ec                mov    %edi,-0x14(%rbp)
 73f:   48 89 75 e0             mov    %rsi,-0x20(%rbp)
 743:   c7 45 f9 57 6f 72 6c    movl   $0x6c726f57,-0x7(%rbp)
 74a:   66 c7 45 fd 64 21       movw   $0x2164,-0x3(%rbp)
 750:   c6 45 ff 00             movb   $0x0,-0x1(%rbp)
 754:   b8 00 00 00 00          mov    $0x0,%eax
 759:   e8 7c ff ff ff          callq  6da <hello>
 75e:   48 8d 45 f9             lea    -0x7(%rbp),%rax
 762:   48 89 c6                mov    %rax,%rsi
 765:   48 8d 3d c3 00 00 00    lea    0xc3(%rip),%rdi
 76c:   b8 00 00 00 00          mov    $0x0,%eax
 771:   e8 3a fe ff ff          callq  5b0 <printf@plt>
 776:   48 8b 45 e0             mov    -0x20(%rbp),%rax
 77a:   48 83 c0 08             add    $0x8,%rax
 77e:   48 8b 00                mov    (%rax),%rax
 781:   48 89 c7                mov    %rax,%rdi
 784:   e8 64 ff ff ff          callq  6ed <buf_copy>
 789:   b8 00 00 00 00          mov    $0x0,%eax
 78e:   c9                      leaveq
 78f:   c3                      retq
```

```
leaveq: mov %rbp, %rsp
        pop %rbp
```

prologue

```
int main(int argc, char
**argv)
{
    char name[]="World!";
    hello();
    printf("Hello %s\n",
name);
    buf_copy(argv[1]);
    return 0;
}
```

x86-64 binary

```
0000000000000734 <main>:
 734:   55                      push   %rbp
 735:   48 89 e5                mov    %rsp,%rbp
 738:   48 83 ec 20             sub    $0x20,%rsp
 73c:   89 7d ec                mov    %edi,-0x14(%rbp)
 73f:   48 89 75 e0             mov    %rsi,-0x20(%rbp)
 743:   c7 45 f9 57 6f 72 6c    movl   $0x6c726f57,-0x7(%rbp)
 74a:   66 c7 45 fd 64 21       movw   $0x2164,-0x3(%rbp)
 750:   c6 45 ff 00             movb   $0x0,-0x1(%rbp)
 754:   b8 00 00 00 00          mov    $0x0,%eax
 759:   e8 7c ff ff ff          callq  6da <hello>
 75e:   48 8d 45 f9             lea    -0x7(%rbp),%rax
 762:   48 89 c6                mov    %rax,%rsi
 765:   48 8d 3d c3 00 00 00    lea    0xc3(%rip),%rdi
 76c:   b8 00 00 00 00          mov    $0x0,%eax
 771:   e8 3a fe ff ff          callq  5b0 <printf@plt>
 776:   48 8b 45 e0             mov    -0x20(%rbp),%rax
 77a:   48 83 c0 08             add    $0x8,%rax
 77e:   48 8b 00                mov    (%rax),%rax
 781:   48 89 c7                mov    %rax,%rdi
 784:   e8 64 ff ff ff          callq  6ed <buf_copy>
 789:   b8 00 00 00 00          mov    $0x0,%eax
 78e:   c9                      leaveq
 78f:   c3                      retq
```

epilogue

```
leaveq: mov %rbp, %rsp
        pop %rbp
```

```
int buf_copy(char* string)
{
    char buffer[50];
    int dummy=200;
    strcpy(buffer, string);
    printf("%d - %s\n", dummy,
buffer);
    return 0;
}
```

x86-64 binary

```
00000000000006ed <buf_copy>:
 6ed:   55                      push   %rbp
 6ee:   48 89 e5                mov    %rsp,%rbp
 6f1:   48 83 ec 50             sub    $0x50,%rsp
 6f5:   48 89 7d b8             mov    %rdi,-0x48(%rbp)
 6f9:   c7 45 fc c8 00 00 00    movl   $0xc8,-0x4(%rbp)
 700:   48 8b 55 b8             mov    -0x48(%rbp),%rdx
 704:   48 8d 45 c0             lea    -0x40(%rbp),%rax
 708:   48 89 d6                mov    %rdx,%rsi
 70b:   48 89 c7                mov    %rax,%rdi
 70e:   e8 7d fe ff ff          callq  590 <strcpy@plt>
 713:   48 8d 55 c0             lea    -0x40(%rbp),%rdx
 717:   8b 45 fc                mov    -0x4(%rbp),%eax
 71a:   89 c6                   mov    %eax,%esi
 71c:   48 8d 3d 03 01 00 00    lea    0x103(%rip),%rdi
 723:   b8 00 00 00 00          mov    $0x0,%eax
 728:   e8 83 fe ff ff          callq  5b0 <printf@plt>
 72d:   b8 00 00 00 00          mov    $0x0,%eax
 732:   c9                      leaveq
 733:   c3                      retq
```

```
int main(int argc, char
**argv)
{
    char name[]="World!";
    hello();
    printf("Hello %s\
n",name);
    buf_copy(argv[1]);
    return 0;
}
```

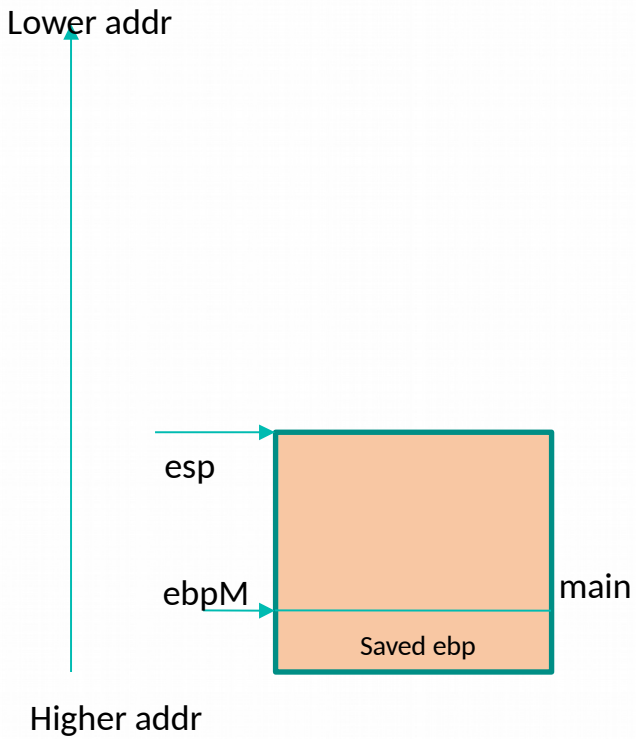
x86-32

```
000005f6 <main>:
 600: 55                    push   %ebp
 601: 89 e5                 mov    %esp,%ebp
 606: 83 ec 1c              sub    $0x1c,%esp
 616: c7 45 e1 57 6f 72 6c  movl   $0x6c726f57,-0x1f(%ebp)
 61d: 66 c7 45 e5 64 21     movw   $0x2164,-0x1b(%ebp)
 623: c6 45 e7 00           movb   $0x0,-0x19(%ebp)
 627: e8 51 ff ff ff        call   57d <hello>
 62c: 83 ec 08              sub    $0x8,%esp
 62f: 8d 45 e1              lea    -0x1f(%ebp),%eax
 632: 50                    push   %eax
 633: 8d 83 3b e7 ff ff     lea    -0x18c5(%ebx),%eax
 639: 50                    push   %eax
 63a: e8 b1 fd ff ff        call   3f0 <printf@plt>
 63f: 83 c4 10              add    $0x10,%esp
 642: 8b 46 04              mov    0x4(%esi),%eax
 645: 83 c0 04              add    $0x4,%eax
 648: 8b 00                 mov    (%eax),%eax
 64a: 83 ec 0c              sub    $0xc,%esp
 64d: 50                    push   %eax
 64e: e8 55 ff ff ff        call   5a8 <buf_copy>
 653: 83 c4 10              add    $0x10,%esp
 656: b8 00 00 00 00        mov    $0x0,%eax
                            mov    %esp,%ebp
 661: 5d                    pop    %ebp
 665: c3                    ret
```

Stack operation

```
M1: PUSH EBP
M2: MOV EBP,ESP
M3: SUB ESP,14
M4: LEA EAX,DWORD PTR SS:
[EBP-14]
M5: PUSH EAX
M6: CALL BoFEx1.00401000
M7: ADD ESP,4
M8: MOV EAX,1
M9:  MOV  ESP, EBP
M10: POP   EBP
M11: RETN
```

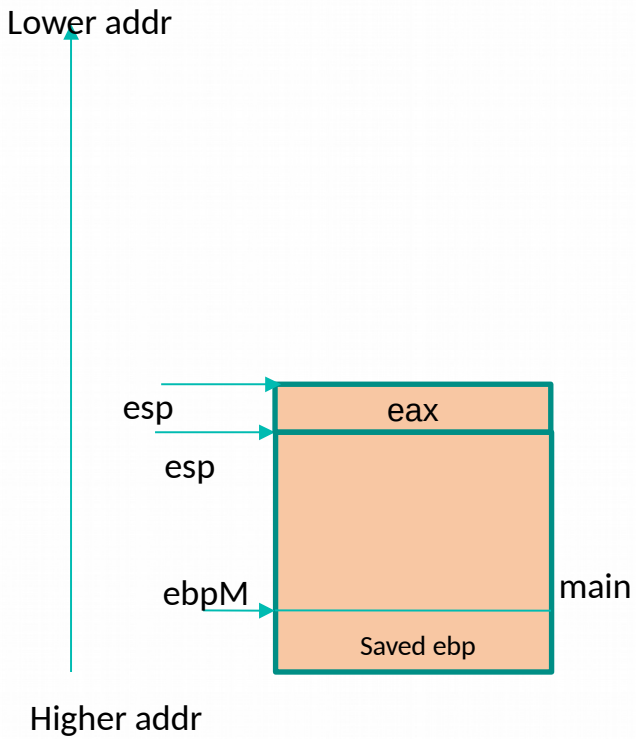
```
PUSH EBP
MOV EBP,ESP
SUB ESP,14
MOV EAX,DWORD PTR SS:[EBP+8]
PUSH EAX
LEA EAX,DWORD PTR SS:[EBP-14]
PUSH EAX
```



Stack operation

```
M1: PUSH EBP
M2: MOV EBP,ESP
M3: SUB ESP,14
M4: LEA EAX,DWORD PTR SS:
[EBP-14]
M5: PUSH EAX
M6: CALL BoFEx1.00401000
M7: ADD ESP,4
M8: MOV EAX,1
M9:  MOV  ESP, EBP
M10: POP   EBP
M11: RETN
```

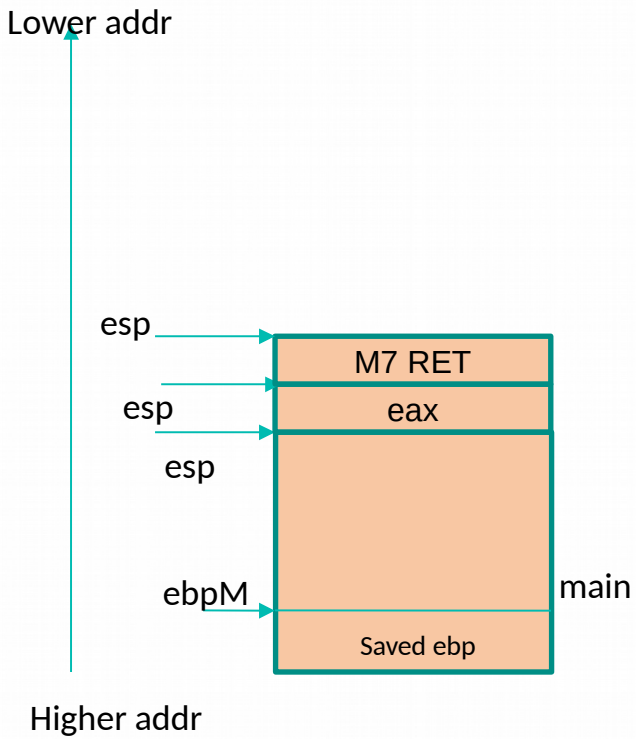
```
PUSH EBP
MOV EBP,ESP
SUB ESP,14
MOV EAX,DWORD PTR SS:[EBP+8]
PUSH EAX
LEA EAX,DWORD PTR SS:[EBP-14]
PUSH EAX
```



Stack operation

```
M1: PUSH EBP
M2: MOV EBP,ESP
M3: SUB ESP,14
M4: LEA EAX,DWORD PTR SS:
[EBP-14]
M5: PUSH EAX
M6: CALL BoFEx1.00401000
M7: ADD ESP,4
M8: MOV EAX,1
M9:  MOV  ESP, EBP
M10: POP   EBP
M11: RETN
```

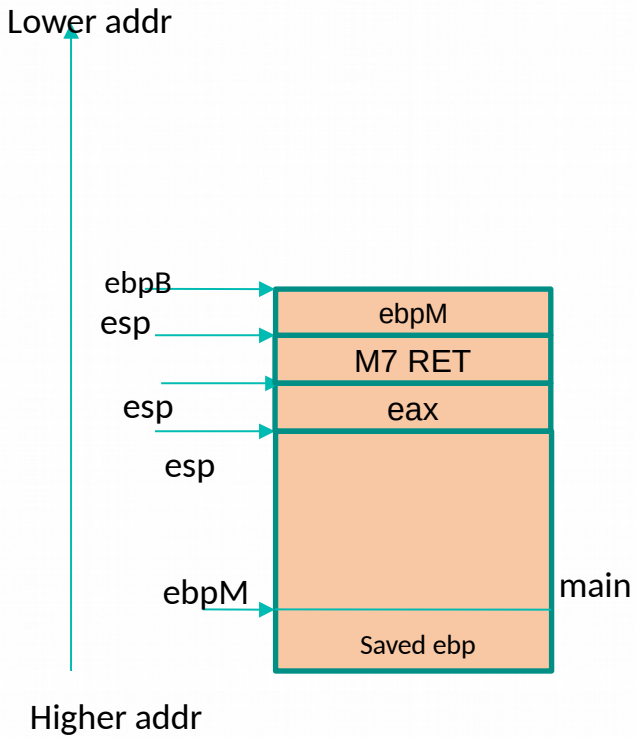
```
PUSH EBP
MOV EBP,ESP
SUB ESP,14
MOV EAX,DWORD PTR SS:[EBP+8]
PUSH EAX
LEA EAX,DWORD PTR SS:[EBP-14]
PUSH EAX
```



Stack operation

```
M1: PUSH EBP
M2: MOV EBP,ESP
M3: SUB ESP,14
M4: LEA EAX,DWORD PTR SS:
[EBP-14]
M5: PUSH EAX
M6: CALL BoFEx1.00401000
M7: ADD ESP,4
M8: MOV EAX,1
M9:  MOV  ESP, EBP
M10: POP   EBP
M11: RETN
```

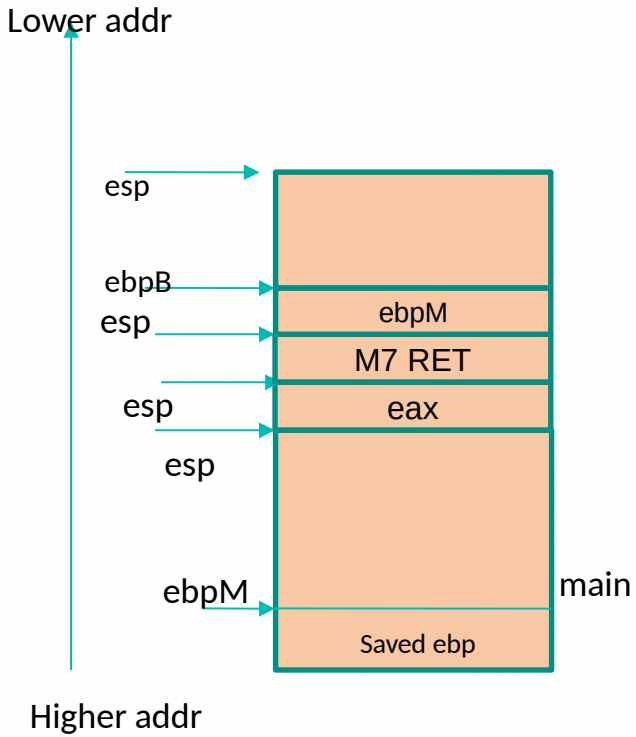
```
PUSH EBP
MOV EBP,ESP
SUB ESP,14
MOV EAX,DWORD PTR SS:[EBP+8]
PUSH EAX
LEA EAX,DWORD PTR SS:[EBP-14]
PUSH EAX
```



Stack operation

```
M1: PUSH EBP
M2: MOV EBP,ESP
M3: SUB ESP,14
M4: LEA EAX,DWORD PTR SS:
[EBP-14]
M5: PUSH EAX
M6: CALL BoFEx1.00401000
M7: ADD ESP,4
M8: MOV EAX,1
M9:  MOV  ESP, EBP
M10: POP   EBP
M11: RETN
```

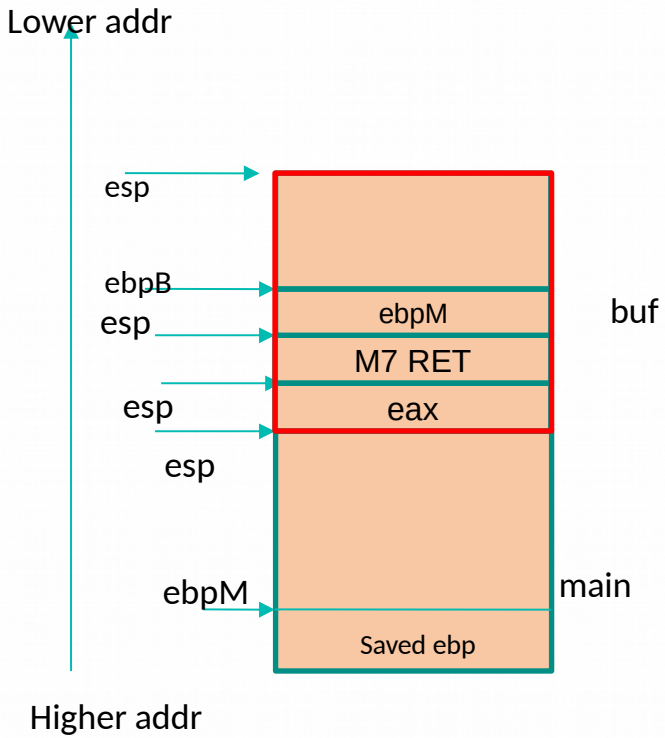
```
PUSH EBP
MOV EBP,ESP
SUB ESP,14
MOV EAX,DWORD PTR SS:[EBP+8]
PUSH EAX
LEA EAX,DWORD PTR SS:[EBP-14]
PUSH EAX
```



Stack operation

```
M1: PUSH EBP
M2: MOV EBP,ESP
M3: SUB ESP,14
M4: LEA EAX,DWORD PTR SS:
[EBP-14]
M5: PUSH EAX
M6: CALL BoFEx1.00401000
M7: ADD ESP,4
M8: MOV EAX,1
M9:  MOV  ESP, EBP
M10: POP   EBP
M11: RETN
```

```
PUSH EBP
MOV EBP,ESP
SUB ESP,14
MOV EAX,DWORD PTR SS:[EBP+8]
PUSH EAX
LEA EAX,DWORD PTR SS:[EBP-14]
PUSH EAX
```



Know your Tools

- Disassembler/debugger
  - Windows: OllyDbg, PyDbg, Immunity Debugger, IDA Pro
  - Linux: GDB, Evan's Debugger (EDB), IDA Pro(!), Ghidra
- Language: C, Python
- Hex viewer: Objdump

compiling options

Useful

- GCC:
  - gcc -fno-stack-protector -z execstack vulnerable.c
    - -fno-stack-protector disables SSP (stack guard)
    - -z execstack marks the stack as executable
  - -m32
    - Producing x86-32 bit code on x86-64 machine
- Windows CL
- /GS-