

The background is a dark, red-tinted image. In the center, there is a silhouette of a person wearing a hood, possibly a hacker or a thief. The background is filled with binary code (0s and 1s) and some faint, illegible text that looks like code or a log. At the bottom, there is a dark, rectangular shape that resembles a computer keyboard.

EXPLOIT DEVELOPMENT

ABOUT ME

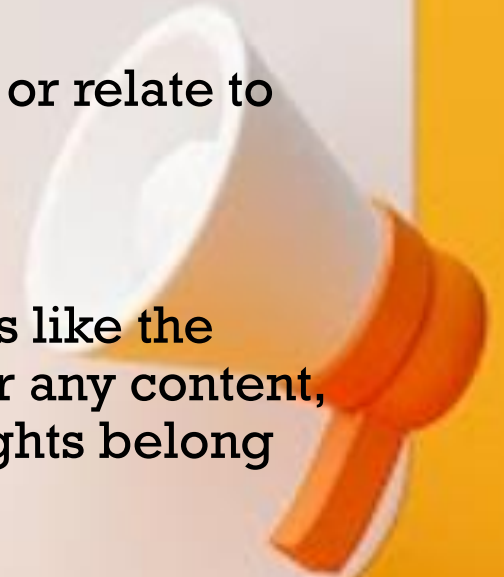
- Manish Sharma (3+yrs experience in AppSec)
- Security Engineer at Victoria's Secret
- eCXD certified | CEH (expired)
- Github/LinkedIn/Twitter: sh377c0d3

\$~: whoami

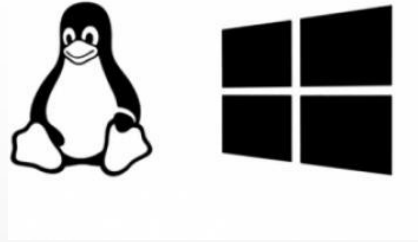


DISCLAIMER

- This entire talk is more at personal level and doesn't contain or relate to any of my former or current professional associations.
- All content has been sourced from publicly available sources like the internet. The presenter does not infringe or claim rights over any content, images and any other work being presented here, and all rights belong to respective owners only.



AGENDA



- Introduction to Exploit Development
- Basic of Windows and Linux Concepts
- Fuzzing and Crash Analysis
- Finding Offset and Overwriting EIP
- Finding Bad Characters
- Stack-based Buffer Overflow



AGENDA (CONTD.)

- Introduction to Egg Hunting
- Return-Oriented Programming (ROP)
- Conclusion and Next Steps



BEFORE WE START

!!!

**CHANGES IN LABS AND
CHALLENGES**



FUN EXAMPLE OF BINARY EXPLOITATION

“[TAS] Super Mario World "Arbitrary Code Execution" in 02:25.19 by Masterjun”



INTRODUCTION TO EXPLOIT DEVELOPMENT

- Exploit development is the process of finding, creating, and developing software or code that takes advantage of a vulnerability in a computer system, network, or application to cause unintended or unauthorized behavior.
- The goal of exploit development is often to gain control of a system, steal sensitive information, or cause damage.

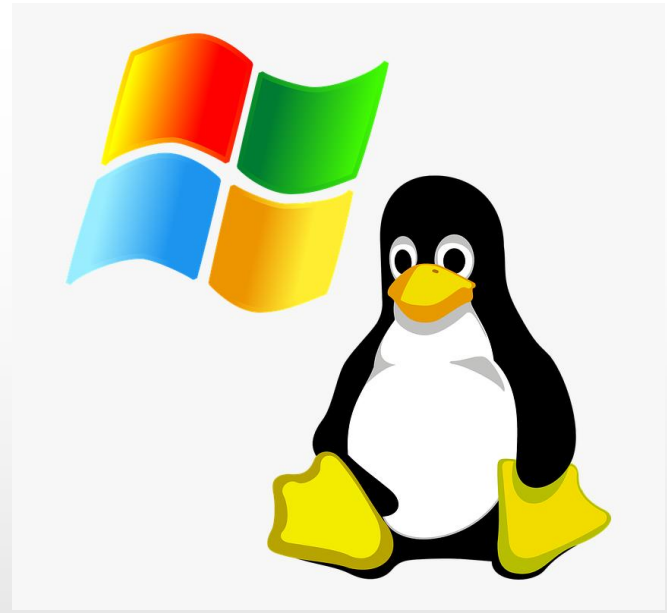


INTRODUCTION TO EXPLOIT DEVELOPMENT

- Identify the Entry Point
- Fuzz the application/software for a crash
- Re-create the crash
- Control the Execution
- Hunt and eliminate bad characters
- Generate shellcode for exploitation
- Obtain a Shell

Base	Top	Size	Rebase	SafeSEH	ASLR	POCompat	OS Dll	Version, ModuleName & Path
0x75b00000	0x75b02000	0x00012000	True	True	False	False	True	6.3.9600.16384 (NtLdr.dll) (C:\Windows\system32\NtLdr.dll)
0x74e10000	0x74e17000	0x00007000	True	True	False	False	True	6.3.9600.16384 (rasadhlp.dll) (C:\Windows\System32\rasadhlp.dll)
0x10000000	0x10027000	0x00027000	False	False	False	False	False	-1.0. [SSLServer32.dll] (C:\EFS Software\Easy Chat Server\SSLServer32.dll)
0x77430000	0x7743e000	0x0000e000	True	True	False	False	True	6.3.9600.16384 (System32\user32.dll) (C:\Windows\SYSTEM32\user32.dll)
0x75430000	0x75439000	0x00009000	True	True	False	False	True	6.3.9600.16384 (CRYPTBASE.dll) (C:\Windows\SYSTEM32\CRYPTBASE.dll)
0x74ba0000	0x74ba0000	0x00000000	True	True	False	False	True	6.3.9600.16384 (ole32.dll) (C:\Windows\SYSTEM32\ole32.dll)
0x75100000	0x75110000	0x00010000	True	True	False	False	True	6.3.9600.16384 (ole32.dll) (C:\Windows\SYSTEM32\ole32.dll)
0x77100000	0x77107000	0x00007000	True	True	False	False	True	6.3.9600.17031 (ntdll.dll) (C:\Windows\SYSTEM32\ntdll.dll)
0x77180000	0x7718e000	0x0000e000	True	True	False	False	True	6.3.9600.16384 (sechost.dll) (C:\Windows\SYSTEM32\sechost.dll)
0x74700000	0x74709000	0x00009000	True	True	False	False	True	6.3.9600.16384 (USERENV.dll) (C:\Windows\SYSTEM32\USERENV.dll)
0x74f00000	0x74f04000	0x00004000	True	True	False	False	True	6.3.9600.16384 (condbase.dll) (C:\Windows\SYSTEM32\condbase.dll)
0x00200000	0x003a1000	0x000d1000	True	False	False	False	False	-1.0. [LIBER32.dll] (C:\EFS Software\Easy Chat Server\LIBER32.dll)
0x74790000	0x74799000	0x00009000	True	True	False	False	True	11.00.9600.16384 (iertutil.dll) (C:\Windows\SYSTEM32\iertutil.dll)
0x74900000	0x74906000	0x00006000	True	True	False	False	True	11.00.9600.16384 (WININET.dll) (C:\Windows\SYSTEM32\WININET.dll)
0x74e20000	0x74e40000	0x00020000	True	True	False	False	True	6.3.9600.16384 (HttpErrors.dll) (C:\Windows\System32\HttpErrors.dll)
0x75a50000	0x75a50000	0x00000000	True	True	False	False	True	6.3.9600.17031 (KERNEL32.DLL) (C:\Windows\SYSTEM32\KERNEL32.DLL)
0x74fc0000	0x74fc0000	0x00000000	True	True	False	False	True	6.3.9600.16384 (MINISET.DLL) (C:\Windows\SYSTEM32\MINISET.DLL)
0x74670000	0x74670000	0x00000000	True	True	False	False	True	6.3.9600.16384 (MIDX32.dll) (C:\Windows\SYSTEM32\MIDX32.dll)
0x75440000	0x75450000	0x00010000	True	True	False	False	True	6.3.9600.16486 (Spool.dll) (C:\Windows\SYSTEM32\Spool.dll)
0x75230000	0x75250000	0x00020000	True	True	False	False	True	6.3.9600.16384 (rsaenh.dll) (C:\Windows\system32\rsaenh.dll)
0x76de0000	0x76de0000	0x00000000	True	True	False	False	True	6.3.9600.16384 (ole32.dll) (C:\Windows\SYSTEM32\ole32.dll)
0x77580000	0x775c1000	0x00041000	True	True	False	False	True	6.3.9600.16384 (SHLDRP.dll) (C:\Windows\SYSTEM32\SHLDRP.dll)
0x75340000	0x75370000	0x00030000	True	True	False	False	True	6.3.9600.16384 (CRYPTSP.dll) (C:\Windows\SYSTEM32\CRYPTSP.dll)
0x75400000	0x7558f000	0x0014f000	True	True	False	False	True	6.3.9600.16384 (USER32.dll) (C:\Windows\SYSTEM32\USER32.dll)
0x76450000	0x76460000	0x00000000	True	True	False	False	True	6.3.9600.16384 (comctl32.dll) (C:\Windows\SYSTEM32\comctl32.dll)
0x75310000	0x75310000	0x00000000	True	True	False	False	True	6.3.9600.16384 (kernel.appcore.dll) (C:\Windows\SYSTEM32\kernel.appcore.dll)
0x74f00000	0x74f00000	0x00000000	True	True	False	False	True	6.3.9600.16384 (IPHLPAPI.DLL) (C:\Windows\SYSTEM32\IPHLPAPI.DLL)
0x750f0000	0x75100000	0x00010000	True	True	False	False	True	6.3.9600.16384 (napinsp.dll) (C:\Windows\system32\napinsp.dll)
0x75120000	0x75120000	0x00000000	True	True	False	False	True	6.3.9600.16384 (uxtheme.dll) (C:\Windows\system32\uxtheme.dll)
0x77200000	0x77217000	0x00017000	True	True	False	False	True	6.3.9600.16384 (OLEAUT32.dll) (C:\Windows\SYSTEM32\OLEAUT32.dll)
0x75200000	0x75200000	0x00000000	True	True	False	False	True	6.3.9600.16384 (profapi.dll) (C:\Windows\SYSTEM32\profapi.dll)
0x75b90000	0x75b90000	0x00000000	True	True	False	False	True	6.3.9600.17031 (SHELL32.dll) (C:\Windows\SYSTEM32\SHELL32.dll)
0x77380000	0x77391000	0x00001000	True	True	False	False	True	6.3.9600.16384 (RPCRT4.dll) (C:\Windows\SYSTEM32\RPCRT4.dll)
0x75080000	0x75070000	0x00070000	True	True	False	False	True	6.3.9600.16384 (RPCRT4.dll) (C:\Windows\SYSTEM32\RPCRT4.dll)
0x75430000	0x75430000	0x00000000	True	True	False	False	True	6.3.9600.17031 (HTTP.dll) (C:\Windows\system32\HTTP.dll)
0x74ff0000	0x74ff0000	0x00000000	True	True	False	False	True	6.3.9600.16384 (winmm.dll) (C:\Windows\System32\winmm.dll)
0x74c20000	0x74c20000	0x00000000	True	True	False	False	True	6.10 (COMCTL32.dll) (C:\Windows\WinSxS\Microsoft.Windows.Common-Controls_65958641
0x75360000	0x7536f000	0x0000f000	True	True	False	False	True	6.3.9600.16384 (TMSXTE.dll) (C:\Windows\SYSTEM32\TMSXTE.dll)
0x74b00000	0x74b07000	0x00007000	True	True	False	False	True	6.3.9600.16384 (OLEPRO32.DLL) (C:\Windows\SYSTEM32\OLEPRO32.DLL)
0x75730000	0x7573f000	0x0000f000	True	True	False	False	True	6.3.9600.17031 (KERNELBASE.dll) (C:\Windows\SYSTEM32\KERNELBASE.dll)
0x75200000	0x75200000	0x00000000	True	True	False	False	True	6.3.9600.16384 (SHCORE.DLL) (C:\Windows\SYSTEM32\SHCORE.DLL)
0x75800000	0x75800000	0x00000000	True	True	False	False	True	6.3.9600.16384 (twainui.dll) (C:\Windows\SYSTEM32\twainui.dll)
0x75340000	0x75340000	0x00000000	True	True	False	False	True	6.3.9600.17031 (GDI32.dll) (C:\Windows\SYSTEM32\GDI32.dll)
0x00400000	0x00400000	0x00000000	False	False	False	False	False	3.1 (EasyChat.exe) (C:\EFS Software\Easy Chat Server\EasyChat.exe)
0x74bc0000	0x74bc0000	0x00000000	True	True	False	False	True	6.3.9600.16384 (MIDX32.dll) (C:\Windows\SYSTEM32\MIDX32.dll)
0x75690000	0x75690000	0x00000000	True	True	False	False	True	6.3.9600.16384 (RPCRT4.dll) (C:\Windows\SYSTEM32\RPCRT4.dll)
0x77670000	0x77677000	0x00007000	True	True	False	False	True	6.3.9600.16384 (NSI.dll) (C:\Windows\SYSTEM32\NSI.dll)
0x77130000	0x77130000	0x00000000	True	True	False	False	True	6.3.9600.16384 (MSG_32.dll) (C:\Windows\SYSTEM32\MSG_32.dll)
0x75300000	0x75300000	0x00000000	True	True	False	False	True	6.3.9600.17031 (bcryptPrimitives.dll) (C:\Windows\SYSTEM32\bcryptPrimitives.dll)





BASIC OF WINDOWS AND LINUX CONCEPTS



STACK



- Stack is an area of memory within a process that is used by the processor to save data.
- Registers are small in size but the fastest among all the temporary data storage, the stack offers a large space.
- Stack is also used to track the execution of the program.

LITTLE BIT OF ASSEMBLY

- All assembly languages are made up of instruction sets
- Instructions are generally simple arithmetic operations that take registers or constant values as arguments
- Also called Operands, OpCode, Op(s), mnemonics
- Intel syntax: operand destination, source
 - `mov eax, 5`
- AT&T syntax: operand source, destination
 - `mov $5, eax`
- We'll be relying on the Intel syntax

Assembly Language

```
mov ecx, ebx
mov esp, edx
mov edx, r9d
mov rax, rdx
```

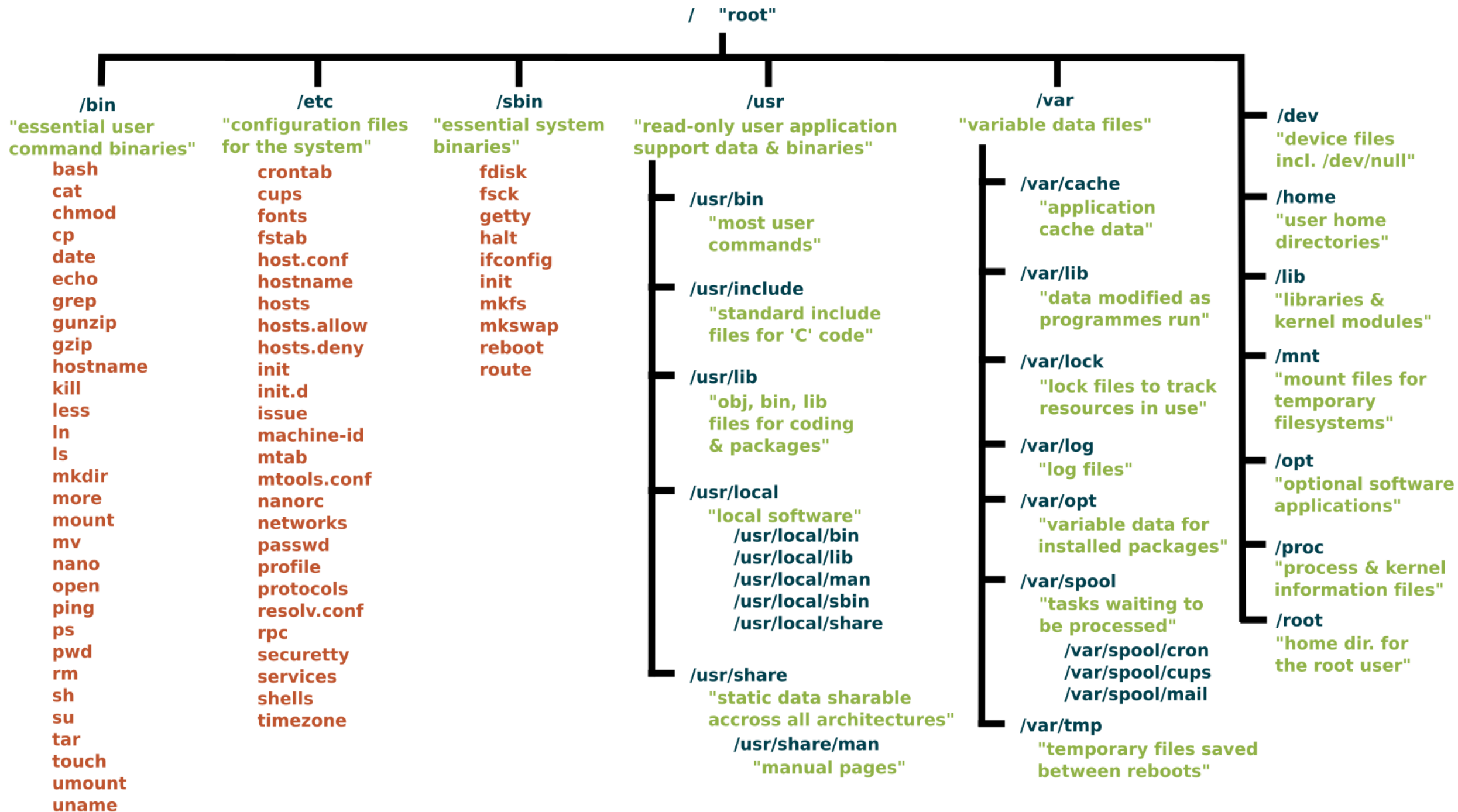
Programmer

Assembler + Linker

Machine Language

```
100101011001
010011111011
111010101101
01010101010
```

Processor

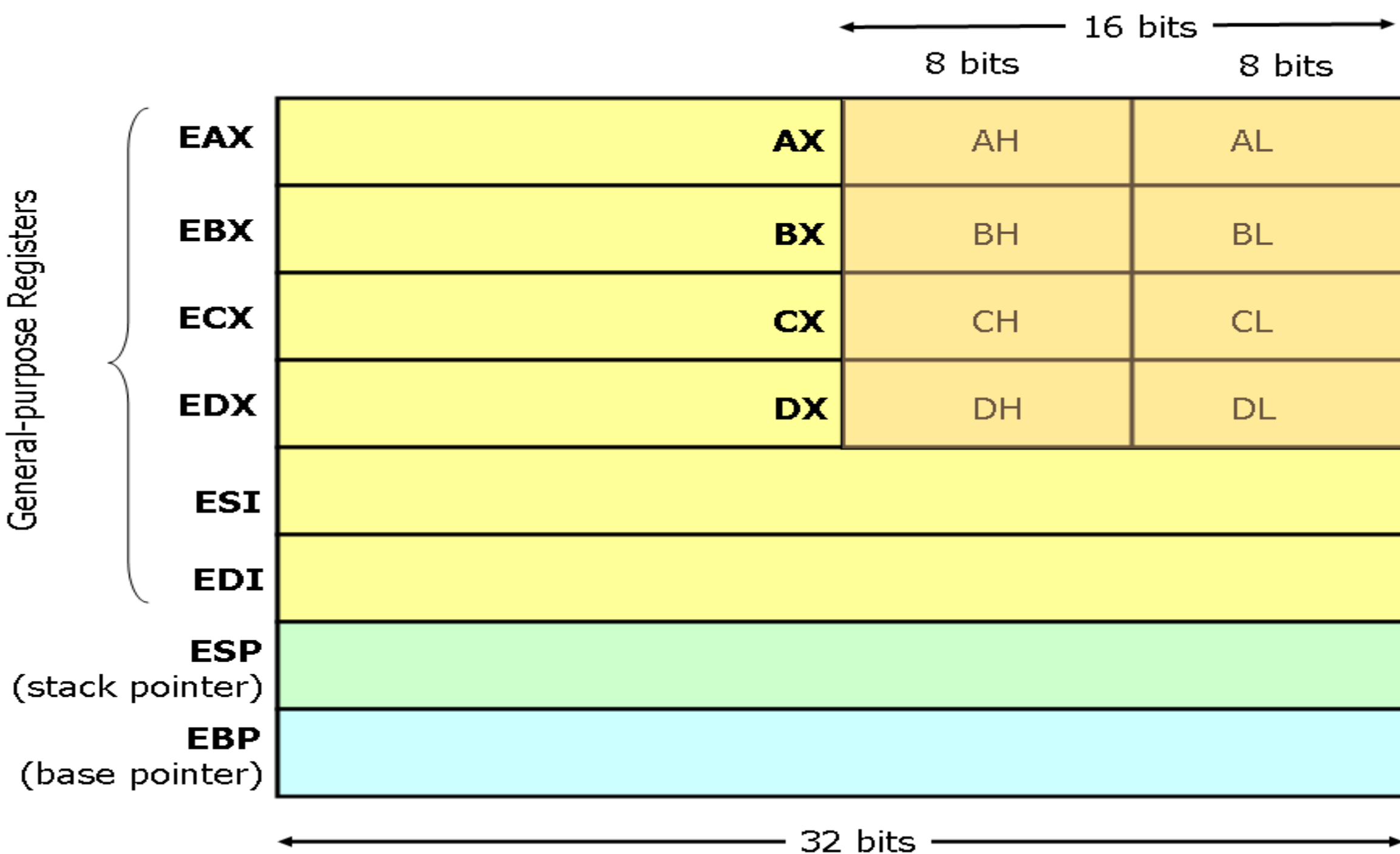


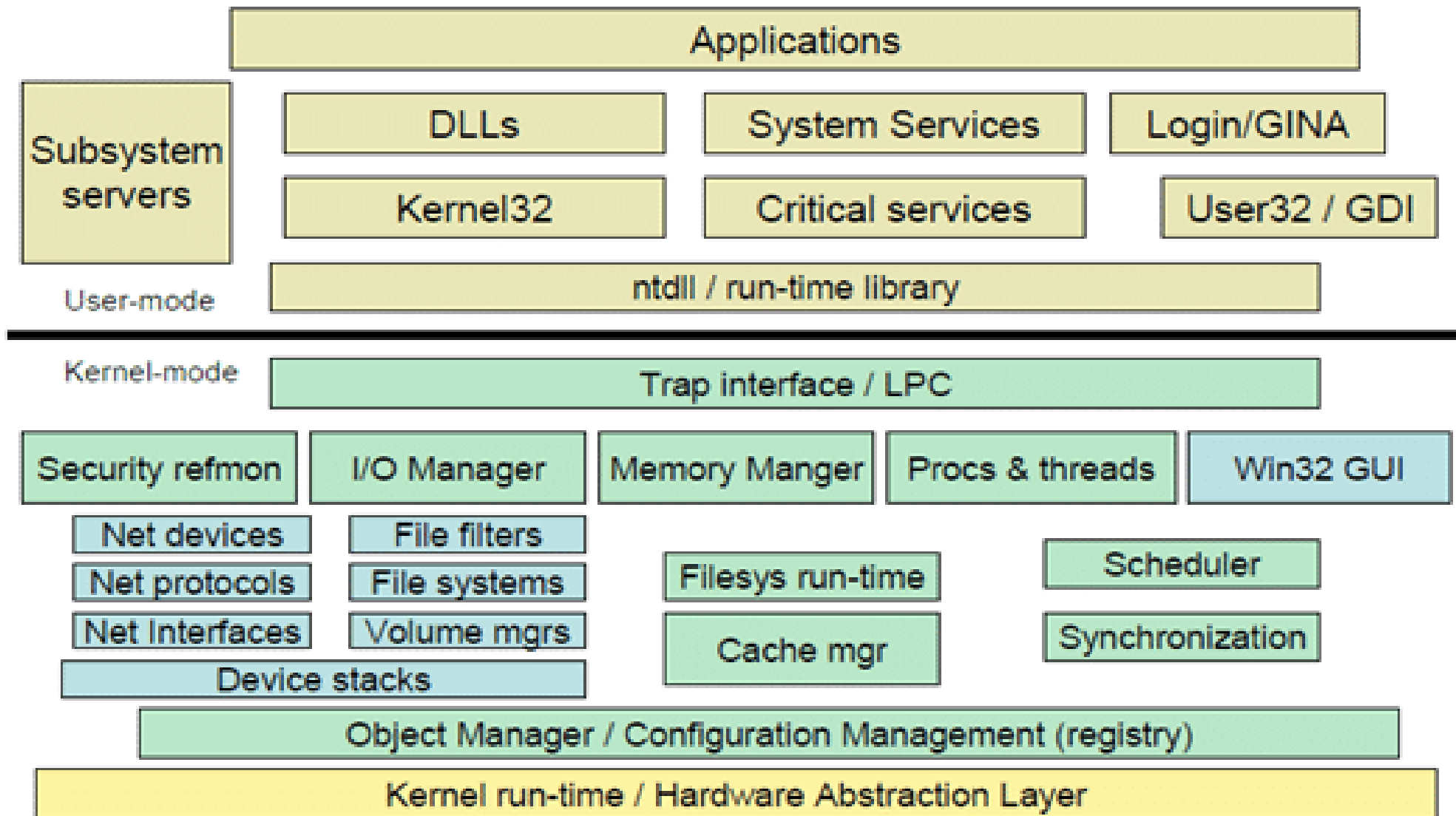
OH! REGISTERS

- EAX EBX ECX EDX - General purpose registers
- ESP - Stack pointer, “top” of the current stack frame (lower memory)
- EBP - Base pointer, “bottom” of the current stack frame (higher memory)
- EIP - Instruction pointer, pointer to the next instruction to be executed by the CPU
- EFLAGS - stores flag bits
- ZF - zero flag, set when result of an operation equals zero
- CF - carry flag, set when the result of an operation is too large/small
- SF - sign flag, set when the result of an operation is negative



General-purpose Registers





FUZZING AND CRASH ANALYSIS

Discovering faults in applications by providing unexpected input and monitoring for exceptions.

Types of fuzzers:

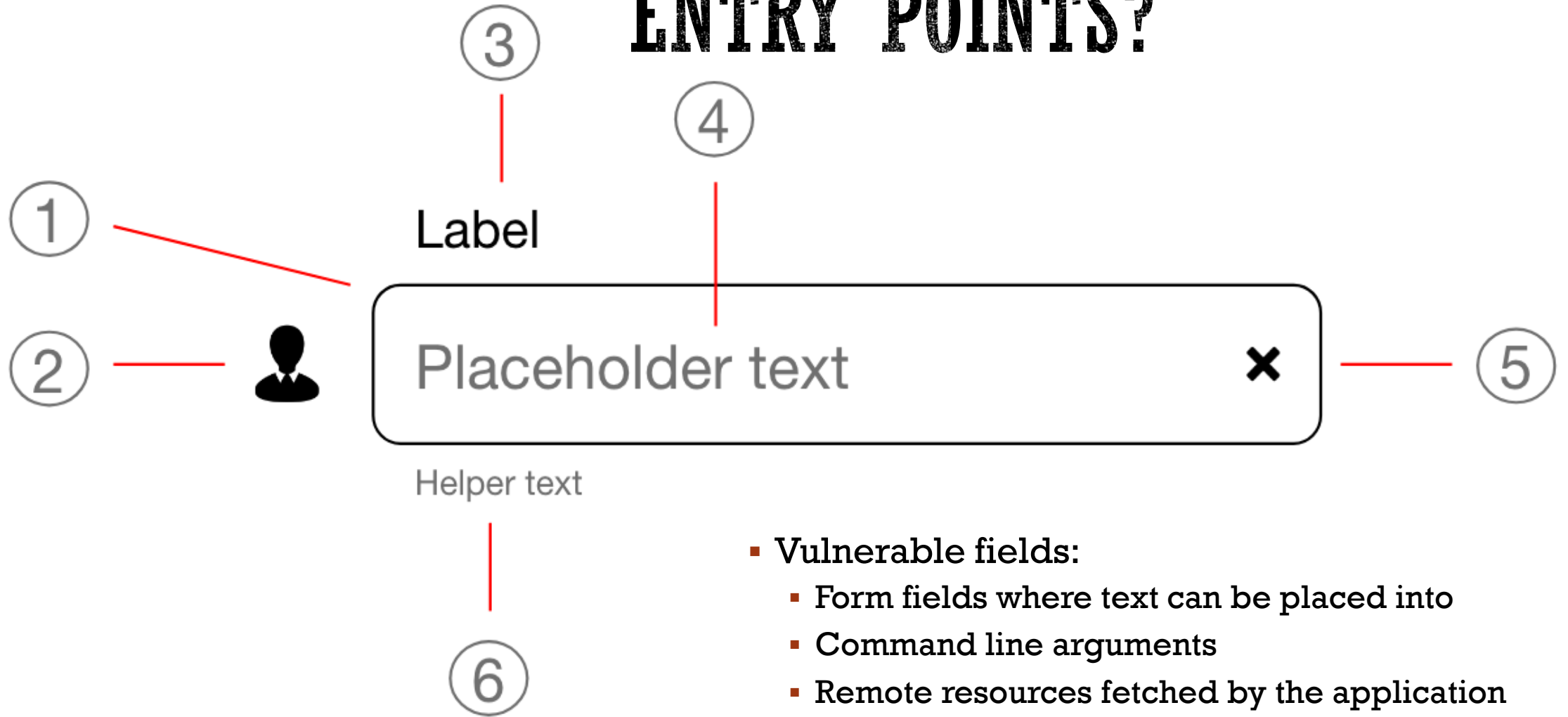
- Mutation-based
- Generation-based

Fuzzing Targets:

- Environment variables and Arguments
- Web application and server
- File Format Network Protocol
- Web browsers
- In-memory



ENTRY POINTS?



- Vulnerable fields:
 - Form fields where text can be placed into
 - Command line arguments
 - Remote resources fetched by the application
 - Files parsed by an application

FUZZING AND CRASH ANALYSIS

- Diverse Input Generation
- Coverage Analysis
- Mutation and Generation
- Boundary Testing & Ethical Considerations
- Continuous Process
- Customization >>> False Positives
- Tool Selection



```
peruvian were-rabbit 2.56b (vulnerable)

process timing
  run time : 0 days, 0 hrs, 8 min, 56 sec
  last new path : 0 days, 0 hrs, 8 min, 55 sec
  last uniq crash : 0 days, 0 hrs, 8 min, 55 sec
  last uniq hang : none seen yet
cycle progress
  now processing : 1 (12.50%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : splice 6
  stage execs : 31/32 (96.88%)
  total execs : 3.72M
  exec speed : 6841/sec
fuzzing strategy yields
  bit flips : 2/1616, 0/1608, 0/1592
  byte flips : 0/202, 0/194, 0/178
  arithmetics : 0/11.3k, 0/3192, 0/767
  known ints : 0/966, 0/4845, 0/7603
  dictionary : 0/0, 0/0, 0/66
  havoc : 0/1.33M, 0/2.35M
  trim : 34.42/69, 0.00%

overall results
  cycles done : 609
  total paths : 8
  uniq crashes : 1
  uniq hangs : 0
map coverage
  map density : 0.02% / 0.05%
  count coverage : 1.03 bits/tuple
findings in depth
  favored paths : 6 (75.00%)
  new edges on : 7 (87.50%)
  new crashes : 97.0k (1 unique)
  total tmouts : 0 (0 unique)
path geometry
  levels : 3
  pending : 0
  pend fav : 0
  own finds : 1
  imported : n/a
  stability : 100.00%

[cpu000: 76%]
```



FINDING OFFSET AND OVERWRITING EIP

- Supply an input of a certain length to the Binary.
- Make the EIP register to point to a certain address.
- EIP control – reuse the dead code
- In Windows, application get access violation in the debugger.
- For Linux, you got “gdb”.
- Creating pattern and finding offset from that pattern is most useful.
- What are the bad characters? Well....
- We got all the things, what now? Now, it's shellcode time!



BEFORE WE MOVE TO FORWARD

Linux

Compile: `gcc -fno-stack-protector -z execstack program.c -o program`

Disable ASLR: `echo 0 | sudo tee /proc/sys/kernel/randomize_va_space`

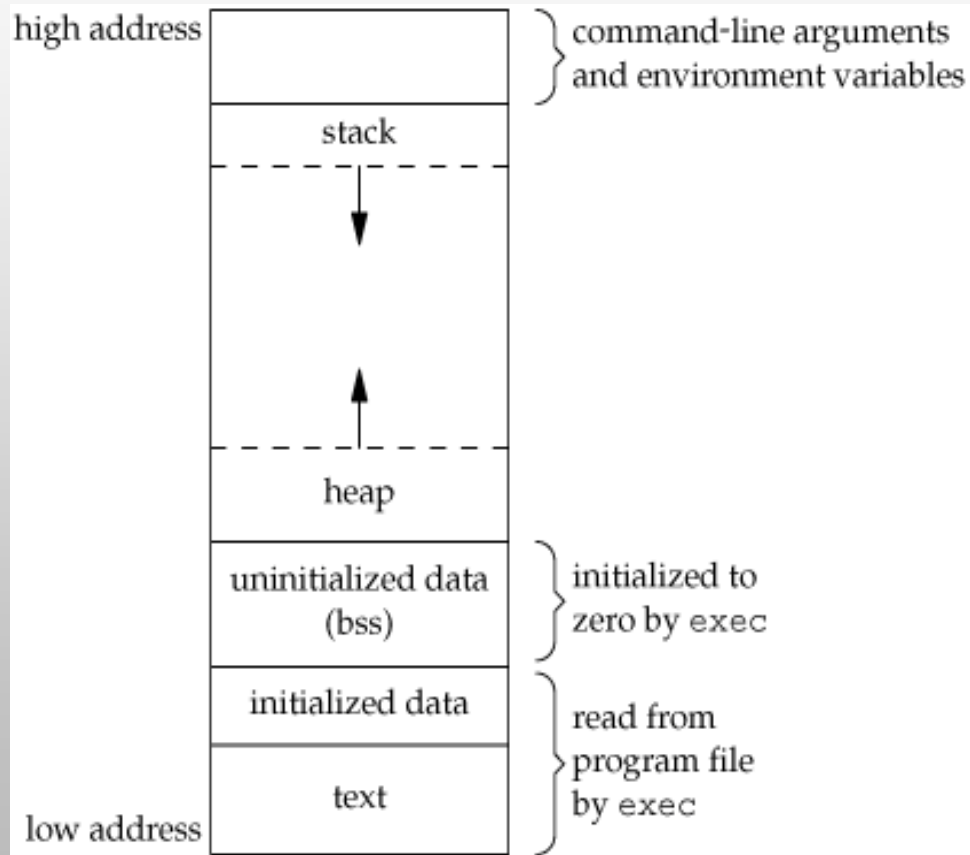


STACK-BASED BUFFER OVERFLOW

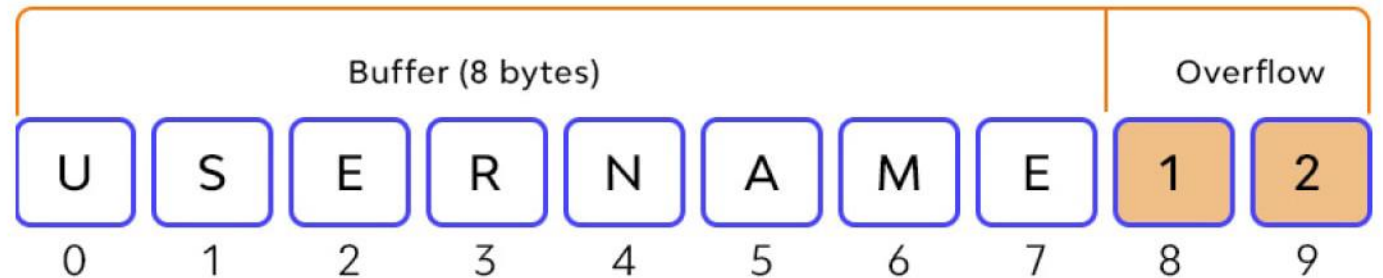
- Stack overflow, also called buffer overflow or stack-based buffer overflow
- It occurs due to a programmatic error.
- This may happen when the program is Insecurely handling user-supplied data.
- The core of buffer overflow exploitation on Windows is the same as it is on Linux.



STACK-BASED BUFFER OVERFLOW



Buffer overflow example



AGAIN !! BEFORE WE MOVE TO HANDS-ON

Linux

Compile: `gcc -fno-stack-protector -z execstack program.c -o program`

Disable ASLR: `echo 0 | sudo tee /proc/sys/kernel/randomize_va_space`



GENERAL PURPOSE REGISTERS

R0 - R6	General purpose
R7	Syscall number
R8 - R10	General purpose
R11	Frame Pointer FP
R12	Intra Procedural IP
R13	Stack Pointer SP
R14	Link Register LR
R15	Program Counter PC

REGISTERS

General purpose registers used for temporary values. R7 stores syscall, used for syscall invocation.

Points to bottom of the Stack Frame, keeps track of boundaries on the stack.

Intra Procedure call scratch Register

Points to bottom of the Stack. Used for allocating space on the Stack.

Receives the return address when a BL or BLX instruction is executed.

Holds the address of the next instruction to be executed.

ENDIANNESS

Little Endian

0x12345678

Byte 0 1 2 3

78 56 34 12

Big Endian

0x12345678

Byte 3 2 1 0

12 34 56 78

ARM 32-BIT

WORD

31 0

MSB

LSB

HALF WORD

15 0

MSB

LSB

BYTE

7 0

MSB

LSB

HIGHER ADDRESS

LOWER ADDRESS

BYTE 0

BYTE 1

BYTE 2

BYTE 3

MSB

LSB

LOAD AND STORE

value at [address] found in Rb is loaded into register Ra

LDR Ra, [Rb]

STR Ra, [Rb]

value found in register Ra is stored to [address] found in Rb

LOAD AND STORE MULTIPLE

value at address found in Ra is loaded into register Rb

LDM Ra, {Rb, Rc}

value found in Rb is stored to address found in Ra

value at address found in Ra (+4) is loaded into register Rc

STM Ra, {Rb, Rc}

value found in Rc is stored to address found in Ra

ARM INSTRUCTIONS

All instructions conditional

32-bit instructions

THUMB INSTRUCTIONS

No conditional instructions

16-bit instructions

BRANCH (B)

SYNTAX

b[cond] label

Diagram showing loop logic with cmp, beq, and bx instructions.

BRANCH & EXCHANGE (BX)

SYNTAX

bx[cond] Rm

Diagram showing branch to register logic with add and bx instructions.

BRANCH & LINK (BL)

SYNTAX

bl[cond] label

Diagram showing branch with link logic with mov, bl, and add instructions.

BRANCH & LINK & EXCHANGE (BLX)

SYNTAX

blx[cond] Rm

Diagram showing branch with link and exchange logic with mov, blx, and add instructions.

CONDITIONAL EXECUTION

Condition Code	Meaning	Flags Tested
EQ	Equal (==)	Z == 1
NE	Not Equal (!=)	Z == 0
GT	Signed >	(Z==0) && (N==V)
LT	Signed <	N != V
GE	Signed >=	N == V
LE	Signed <=	(Z==1) (N!=V)
CS or HS	U. Higher or Same	C == 1
CC or LO	U. Lower	C == 0
MI	Negative -	N == 1
PL	Positive +	N == 0
AL	Always executed	-
NV	Never executed	-
VS	S. Overflow	V == 1
VC	No Overflow	V == 0
HI	U. Higher	(C==1) && (Z==0)
LS	U. Lower or same	(C==0) (Z==0)

CPSR / APSR

(Current Program Status Register)

N	Z	C	V	Q	J	GE	E	A	I	F	T	M
---	---	---	---	---	---	----	---	---	---	---	---	---

Cmp/Test Instructions

CMN (compare), CMP (compare negative), TEQ (test equivalence), TST (test bits)

always update flags only if suffix Registers unchanged.

Other instructions, like MOVN (move, update flag), ADDS (add, update flag), SUBS (subtract, update flag), [...]

Example: CMP & LT

mov r0, #2
mov r1, #4
cmp r0, r1
movlt r2, #8

r0 = 2
r1 = 4
r0 - r1 = 2 - 4 = -2
(N != V) == true
(N = V) == false

set negative flag

Flag status: N=1, Z=0, C=0, V=0

Example: CMP & EQ

mov r0, #2
mov r1, #2
cmp r0, r1
beq func1
mov r2, #8

r0 = 2
r1 = 2
r0 - r1 = 2 - 2 = 0
(Z == 1) == true

set zero flag

Flag status: N=0, Z=1, C=0, V=0

“\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f”

“\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40”

“\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f”

“\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f”

“\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f”

“\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\b3\b4\b5\b6\b7\b8\b9\xba\xbb\xbc\xbd\xbe\xbf”

“\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xdd\xde\xdf”

“\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xfa\xfb\xfc\xfd\xfe\xff”

AN ADVENTURE TOO BIG FOR THE REAL WORLD



A STEVEN SPIELBERG FILM

READY PLAYER ONE

MUSIC BY ALAN SILVESTRI
EXECUTIVE PRODUCERS ADAM BOITMAN, DANIEL LUPI, CHRIS DEFARIA AND BRUCE BERTMAN
BASED ON THE NOVEL BY ERNEST CLINE
SCREENPLAY BY ZAH PENNING
DIRECTED BY STEVEN SPIELBERG
BY DONALDO DE LINE, P.G.A. COSTUME DESIGNER KRISTIE MACOSKO KRIEGER, P.G.A. EDITOR DAN FARAH, P.G.A.

INTRODUCTION TO EGG HUNTING

- Egg Hunter shellcode simply means small sized shellcode
- Writing shellcode to Exploit within a Limited space
- Shellcode won't fit in the available space
- Storing User input in the memory for long run than expected.
- Relays on system calls that have ability to traverse process memory



INTRODUCTION TO EGG HUNTING

- Character transformation may occur
- These are not only limited to characters
- Sometime whole memory chunk
- Reason?? It's Unknown
- Sometimes the buffer is truncated -- hard to fit shellcode
- Then how to archive exploitation?



INTRODUCTION TO EGG HUNTING

- Egg Hunter can be generated in Immunity Debugger with the help of Mona.py
 - `!mona egg -t r00t3r`
 - Simple Format of an Egg Hunter shell is:
 - EGGEGG + shellcode
- Here EGGEGG is nothing, but tag or word repeated twice

So, Step goes like this:

1. Write a shellcode in the limited buffer to find EGGEGG
2. Once the shellcode is executed, then it'll look for both occurrence of EGG
3. Once EGGEGG is found it'll execute our desired exploit which is present after EGGEGG!



Immunity Debugger 1.85.0.0 : R*lyeh
Need support? visit <http://forum.immunityinc.com/>
"C:\Documents and Settings\Administrator\Desktop\UuInServer\vuInserver.exe"

Console file 'C:\Documents and Settings\Administrator\Desktop\UuInServer\vuInserver.exe'

[22:10:22] New process with ID 0000069C created

00401130 Main thread with ID 00000698 created

00400000 Modules C:\Documents and Settings\Administrator\Desktop\UuInServer\vuInserver.exe

62500000 Modules C:\Documents and Settings\Administrator\Desktop\UuInServer\essfunc.dll

71AA0000 Modules C:\WINDOWS\system32\WS2HELP.dll

71AB0000 Modules C:\WINDOWS\system32\WS2_32.DLL

77C10000 Modules C:\WINDOWS\system32\msvcrt.dll

77DD0000 Modules C:\WINDOWS\system32\ADVAPI32.dll

77E70000 Modules C:\WINDOWS\system32\RPCRT4.dll

77FE0000 Modules C:\WINDOWS\system32\Secur32.dll

7C800000 Modules C:\WINDOWS\system32\kernel32.dll

7C900000 Modules C:\WINDOWS\system32\ntdll.dll

00401130 [22:10:22] Program entry point

71A50000 Modules C:\WINDOWS\system32\mswsock.dll

662B0000 Modules C:\WINDOWS\system32\hnetofg.dll

77F10000 Modules C:\WINDOWS\system32\GDI32.dll

7E410000 Modules C:\WINDOWS\system32\USER32.dll

0BADF000 [+] Command used:

0BADF000 !mona egg -t sh377c0d3

0BADF000 [+] Egg set to w00t

0BADF000 [+] Generating traditional 32bit egghunter code

0BADF000 [+] Preparing output file 'egghunter.txt'

0BADF000 - (Re)setting logfile egghunter.txt

0BADF000 [+] Egghunter (32 bytes):

"\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xod\x2e\x3c\x05\x5a\x74"

"\xef\x87\x30\x30\x74\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7"

0BADF000

0BADF000 [+] This mona.py action took 0:00:00.016000

!mona egg -t sh377c0d3|



BEFORE WE GET INTO ROP

DEFENCE :

ASLR, NX, DEP, STACK CANARY ... and more

<https://github.com/sashs/Ropper>

<https://github.com/JonathanSalwan/ROPgadget>

<https://github.com/corelan/mona>



RETURN-ORIENTED PROGRAMMING (ROP)

- There is only ASLR, you could brute force the shellcode address.
- Only NX, you could return to libc , as it is always at the same address.
- What if ... there is ASLR + NX?
 - Can't brute force now !
 - Can't return to the system, as it will always be at a different address.
 - Now How to Achieve Exploitation ?!



RETURN-ORIENTED PROGRAMMING (ROP)

- Return-Oriented Programming is successor of return-to-libc attack technique.
- In return-oriented programming, you can chain multiple functions to form a ROP chain.
- Gadgets? These are nothing but sequence of code residing in executable memory followed by return instruction.



RETURN-ORIENTED PROGRAMMING (ROP)



- Abuse code that is:
 - Already within the process address space
 - Not randomized (remember that ASLR randomizes certain sections, not everything)
- There could be another function instead of gadget().
- The only thing that should be done is that the stack should be prepared for another function.



RETURN-ORIENTED PROGRAMMING (ROP)

Gadgets and Returns

- 1122aa33 holds the real, intended instruction
- Let's offset it 1 byte and now it points to 1122aa34
- Just 1 byte off and completely different instructions followed by a return!
- This is how gadgets are built !!!

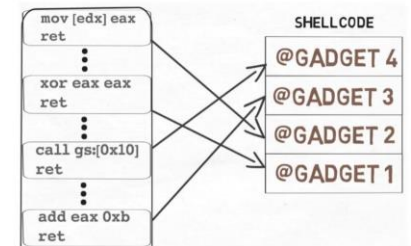
- **mov EIP, [ESP]**
- **add esp, 4** //those two are the standard RET implementation
- **add esp, 4** //this is the 4 (of RET 4) – align the stack by 4 bytes. If it's ret 8, then the following will be added: **esp, 8**



RETURN-ORIENTED PROGRAMMING (ROP)

- Explore and achieve Overflow vulnerability
- Overwrite return address program with a ROP gadget
- ROP gadget pop a value from stack and store it in register
- Now find out another ROP gadget for a specific function
- Chain gadgets to pop value into a REGISTER
- Execute second gadget to perform another specific operation
- BOOM !! ROP chain is executed calling vulnerable function!

Return Oriented Programming



[NX bypass with mprotect\(\)](#)





THANK YOU

