บจก.สยามถนัดแฮก

NCSA CTF Boot Camp #2

Binary Exploitation (Pwn) & Binary Reverse Engineering

Responsible: Mr. Pichaya Morimoto
Version (Date): 1.0 (2024-09-14)
Confidentiality class: Public

whoami

Pichaya (LongCat) Morimoto
Lead Penetration Tester
Siam Thanat Hack Co., Ltd.

Peeratach (Peter) Butto
Penetration Tester
Siam Thanat Hack Co., Ltd.

Yasinthon (Not) Khemprakhon
Penetration Tester
Siam Thanat Hack Co., Ltd.

Disclaimer

- จุดประสงค์ของการบรรยาย นี้เพื่อแบ่งปันความรู้ ทางด้านความปลอดภัยระบบสารสนเทศ
- ไม่สนับสนุนการนำความรู้ทางด้านความปลอดภัยฯ ไปใช้ในทางที่ผิดกฎหมายทั้งหมด
- ตัวอย่างโค้ด และรูปใน การบรรยาย นี้ เป็นระบบจำลองของทางผู้บรรยาย ไม่ใช่ระบบลูกค้า

Agenda (Day 1)

เวลา	รายละเอียด
09.15 - 09.45	ความรู้เบื้องต้นเกี่ยวกับ CTF
09.45 - 10.30	Network Security
10.30 - 10.45	พักเบรก
10.45 - 12.00	Web Application Security
12.00 - 13.00	พักรับประทาน อาหารกลางวัน
13.00 - 14.30	Digital Forensics
14.30 - 14.45	พักเบรก
14.45 - 16.00	Pwnable & Reverse Engineering
16.00 - 18.00	เข้าห้องพัก
18.00 - 19.00	รับประทานอาหารเย็น
19.00 - 21.00	เสวนาแนะแนวเส้นทางอาชีพ

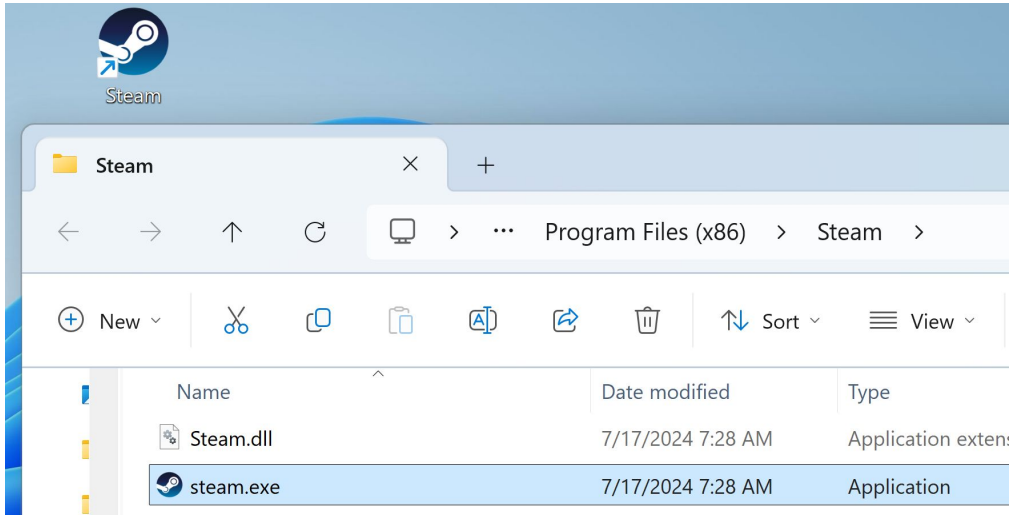
Content Overview

- Binary Reverse Engineering คืออะไร
  - Binary File Format (ELF,PE)
  - De/Compilation และ Disassemble
  - ภาษา Assembly
  - Debugging
- System Exploitation (Pwn) คืออะไร
  - Memory Layout
  - Format String
  - Shellcode
- ลองทำแลป !
  - Lab 1: Vanilla Format String
  - Lab 2: Vanilla Heap Overflow

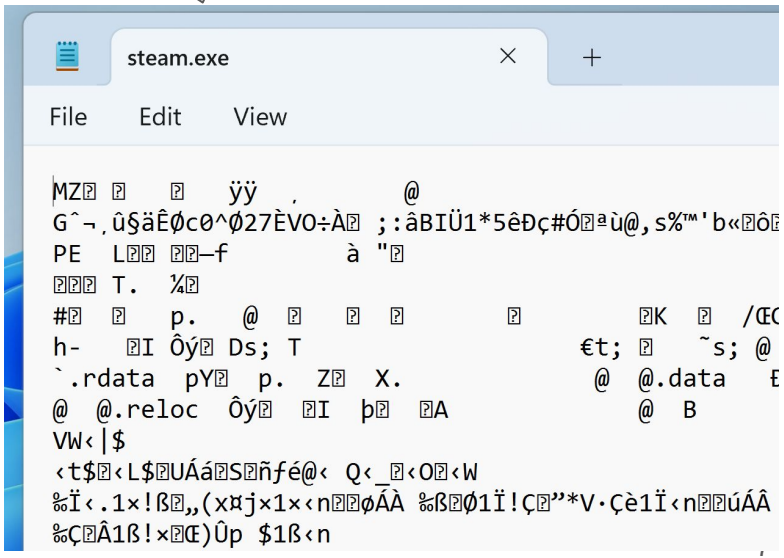
Binary Reverse Engineering

Binary Reverse Engineering คืออะไร? - Binary คืออะไร

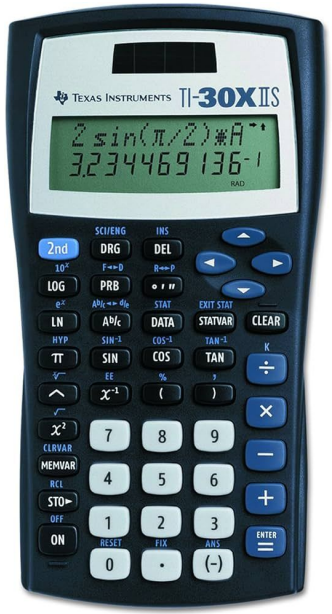
ไฟล์ Binary ของโปรแกรม
- Windows = PE (.exe)
- Linux = ELF



เนื้อหาไฟล์ เป็น Unicode
ที่อ่านเป็นภาษามนุษย์ไม่รู้เรื่อง

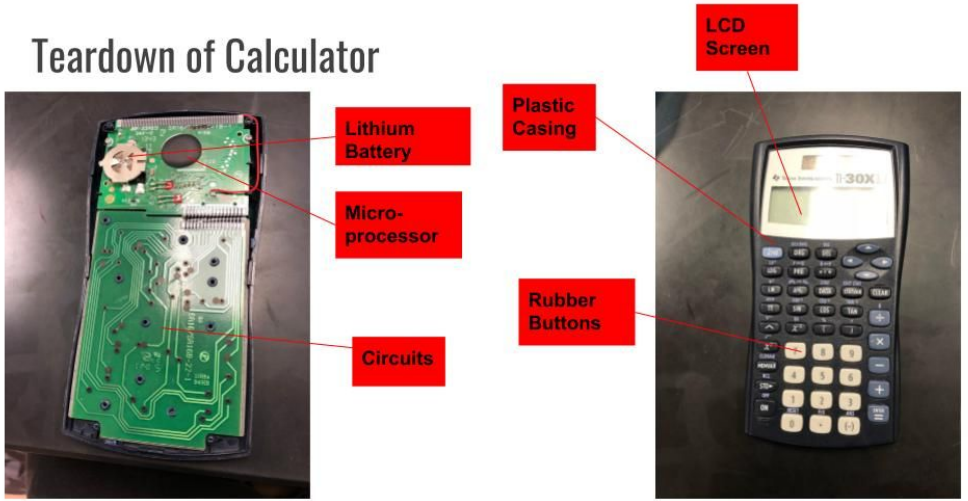


Binary Reverse Engineering คืออะไร? - Reverse Engineering คืออะไร (1/2)



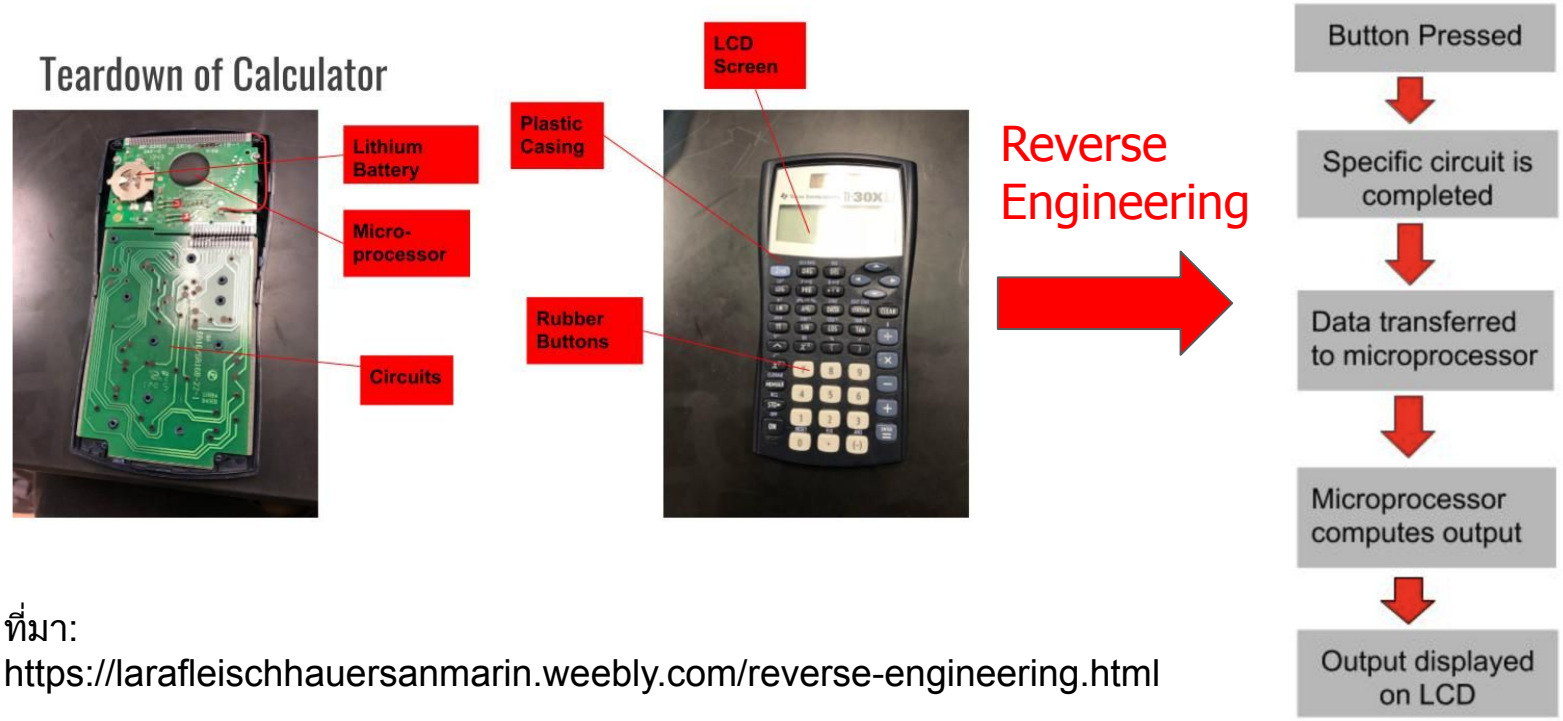
Engineered Product

Reverse Engineering



ที่มา: https://larafleischhauersanmarin.weebly.com/reverse-engineering.html

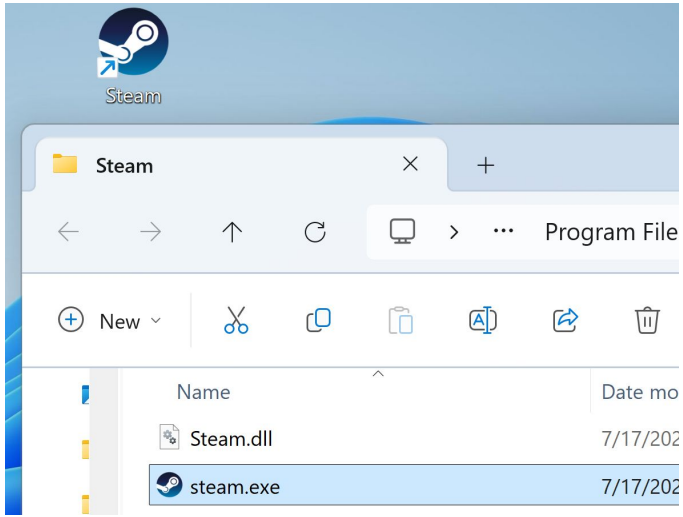
Binary Reverse Engineering คืออะไร? - Reverse Engineering คืออะไร (2/2)



ที่มา:
https://larafleischhauersanmarin.weebly.com/reverse-engineering.html

Binary Reverse Engineering คืออะไร?

ไฟล์ Binary ของโปรแกรม
- Windows = PE (.exe)
- Linux = ELF



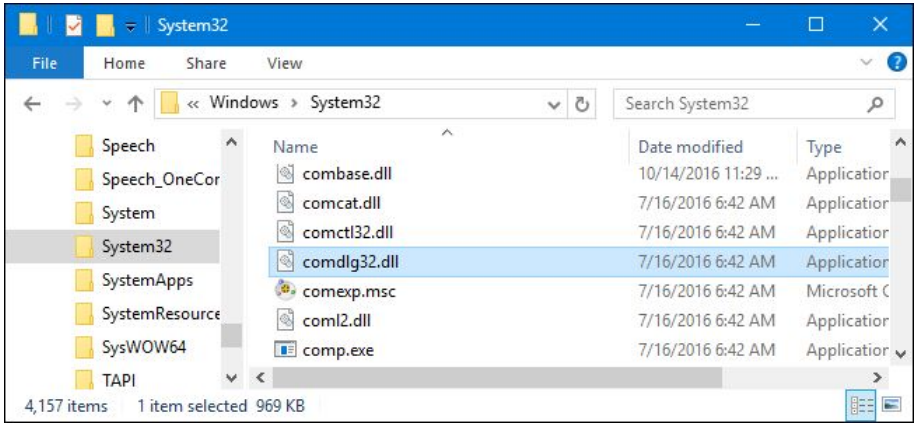
Reverse Engineering

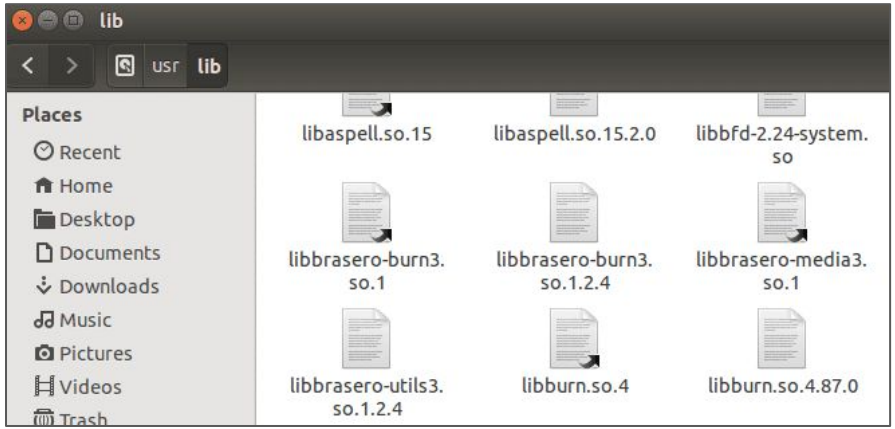
การทำความเข้าใจไฟล์ Binary ของโปรแกรม (โดยมักจะไม่มีโค้ด) เพื่อจุดประสงค์

- [ดี] Malware Analysis
- [ดี] Vulnerability Research
- [ไม่ดี] Software Cracking
- [ไม่ดี] Game Cheating
- [ดี] Fun & Profit

เปรียบเทียบนามสกุลไฟล์ Binary ของ Windows และ Linux

	Windows	Linux
Progam Binary	.exe	ไม่มีนามสกุลไฟล์ (ELF)
Static Library	.lib	.a
Shared Library	.dll	.so





กระบวนการ Binary Reverse Engineering

Number System (เลขฐาน)

Binary ฐาน 2 | Decimal ฐาน 10 | Hex ฐาน 16 | Oct ฐาน 8

Base 2	Base 10	Base 16	Base 8
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	8	10
1001	9	9	11
1010	10	A	12
1011	11	B	13
1100	12	C	14
1101	13	D	15
1110	14	E	16
1111	15	F	17

www.trance-cat.com

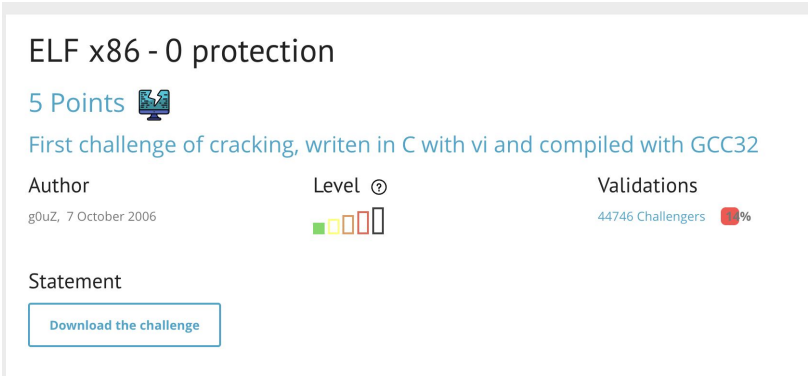
ที่มา: https://www.trance-cat.com/electrical-circuit-calculators/en/base-2-base-16-base-36-calculator.php

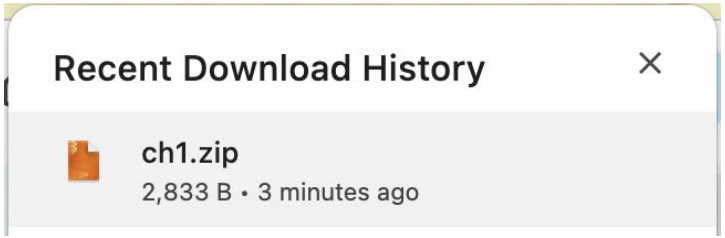
ASCII Table

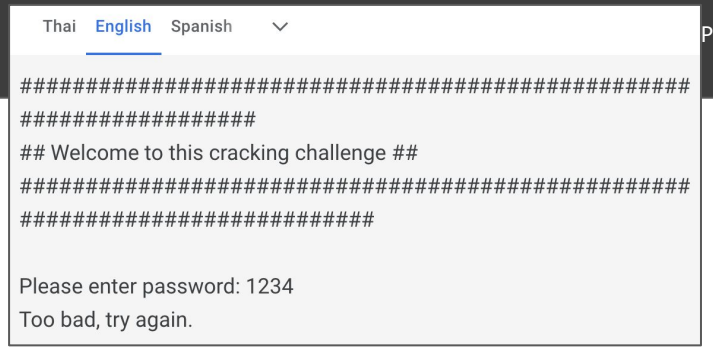
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

https://simple.wikipedia.org/wiki/ASCII

ตัวอย่างโจทย์ CTF - ELF x86 - 0 protection







https://www.root-me.org/en/Challenges/Cracking/ELF-x86-0-protection

ตัวอย่างโจทย์ CTF - ELF x86 - 0 protection

หารหัสผ่าน
หา Flag

เช็ครหัสผ่านยังไง?
เก็บรหัสผ่านยังไง?

https://www.root-me.org/en/Challenges/Cracking/ELF-x86-0-protection

ตัวอย่างโจทย์ CTF - ELF x86 - 0 protection

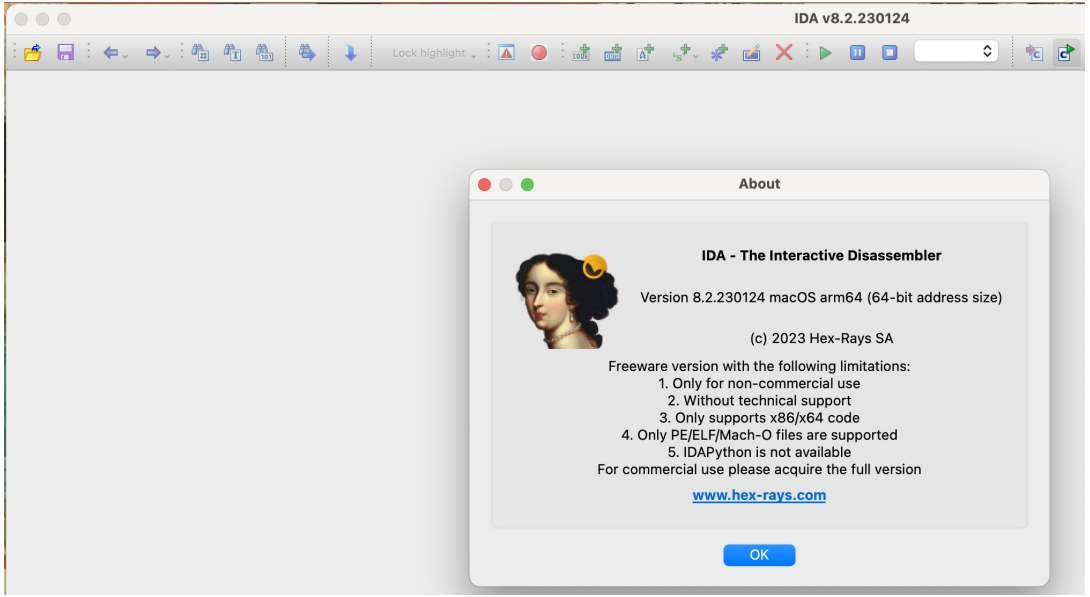
***** Sploitler Alert *****
รหัสผ่านอยู่ในไฟล์ Binary นั้นแหละ

https://www.root-me.org/en/Challenges/Cracking/ELF-x86-0-protection

$ strings ./ch1.bin

```
/vbox_shared/ctf/root-me > strings ./ch1.bin
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
puts
realloc
getchar
__errno_location
malloc
stderr
fprintf
strcmp
strerror
__libc_start_main
GLIBC_2.0
PTRh@
[^_]
%s : "%s"
Allocating memory
Reallocating memory
123456789
##############################################################
##        Bienvennue dans ce challenge de cracking          ##
##############################################################
Veuillez entrer le mot de passe :
Bien joue, vous pouvez valider l'epreuve avec le pass : %s!
Dommage, essaye encore une fois.
```

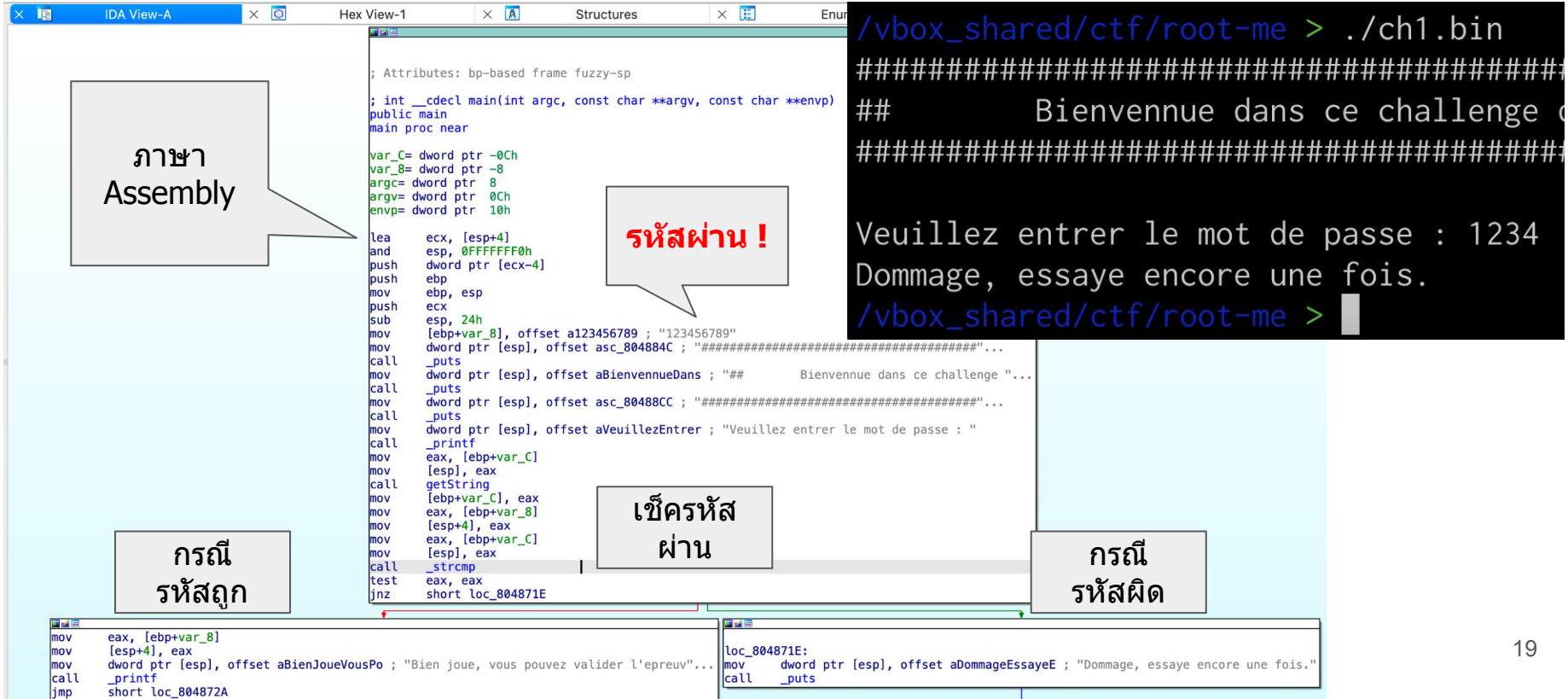
ตัวอย่างโจทย์ CTF - ELF x86 - 0 protection



- IDA ย่อมาจาก Interactive Disassembler
- เครื่องมือแปลง Machine Code เป็นโค้ดที่มนุษย์อ่านเข้าใจได้
- ใช้ในการตรวจสอบและวิเคราะห์ช่องโหว่ในโปรแกรม
- นิยมในงาน Reverse Engineering และความปลอดภัย

https://hex-rays.com/ida-free/

ตัวอย่างโจทย์ CTF - ELF x86 - 0 protection



ตัวอย่างโจทย์ CTF - ELF x86 - 0 protection

หารหัสผ่าน

เช็ครหัสผ่านยังไง?
เก็บรหัสผ่านยังไง?

```
/vbox_shared/ctf/root-me > ./ch1.bin
############################################################
##        Bienvennue dans ce challenge de cracking        ##
############################################################

Veuillez entrer le mot de passe : 123456789
Bien joue, vous pouvez valider l'epreuve avec le pass : 123456789!
```

ได้ Flag คือ
123456789!

https://www.root-me.org/en/Challenges/Cracking/ELF-x86-0-protection

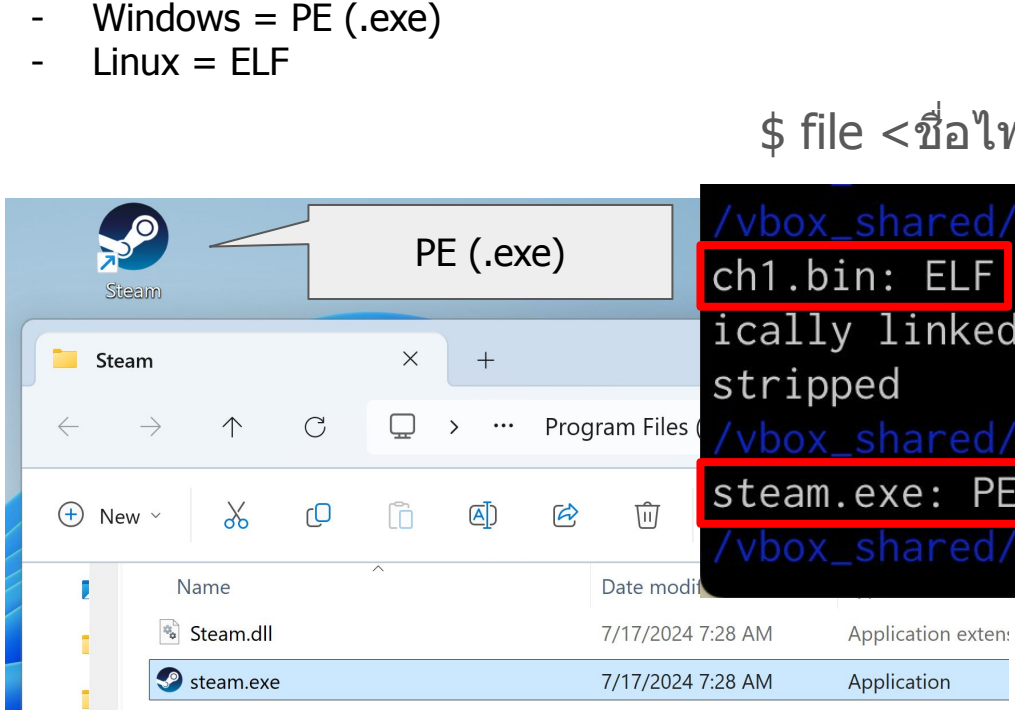
Binary File Format (ELF,PE)

ไฟล์ Binary ของโปรแกรม

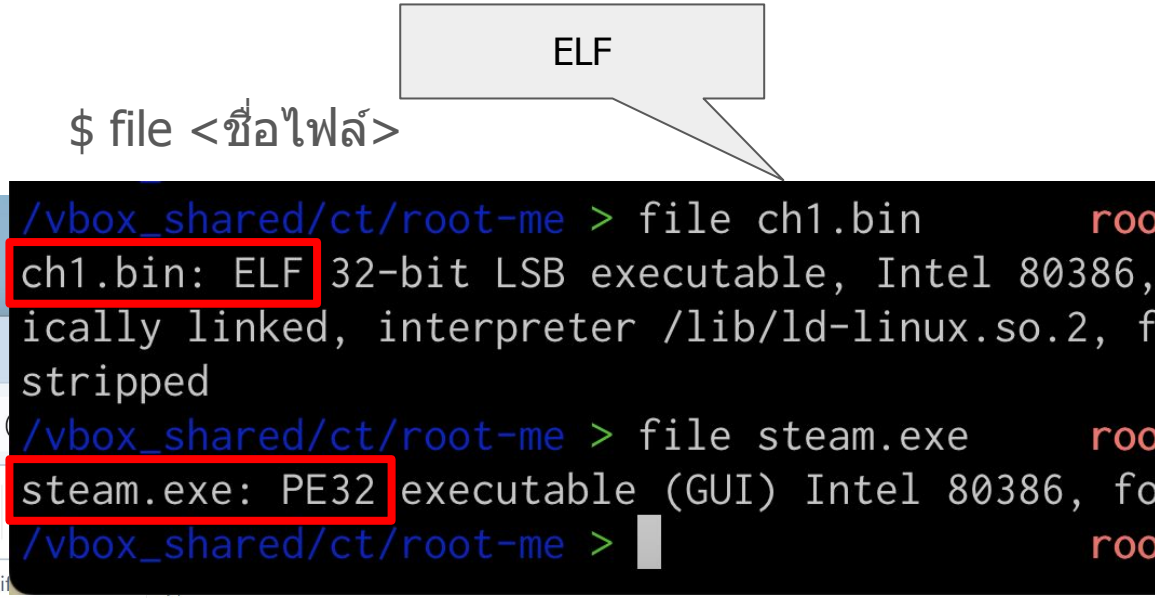
- Windows = PE (.exe)
- Linux = ELF

คำอธิบาย

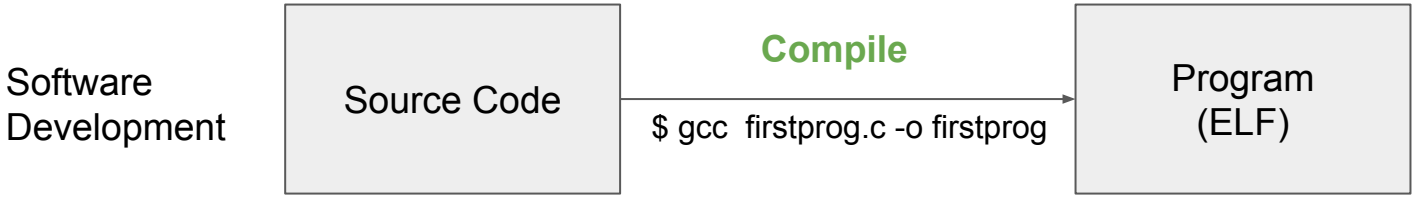
ELF (Executable and Linkable Format): เป็นรูปแบบไฟล์ปฏิบัติการที่ใช้ในระบบปฏิบัติการตระกูล Linux และ Unix
PE (Portable Executable): เป็นรูปแบบไฟล์ปฏิบัติการที่ใช้ในระบบปฏิบัติการ Windows



$ file <ชื่อไฟล์>



Compilation vs Decompilation vs Disassemble



คำอธิบาย

Compilation: การแปลงโค้ดจากภาษาที่มนุษย์เขียน (เช่น C, Java) ให้เป็น **Machine Code** เพื่อให้คอมพิวเตอร์เข้าใจและรันได้

File: firstprog.c

```c
#include <stdio.h>

int main()
{
        int i;
        for(i=0; i < 10; i++) // Loop 10 times.
        {
                puts("Hello, world!\n"); // put the string to the output.
        }
        return 0; // Tell OS the program exited without errors.
}
```

./firstprog

```
┌──(root㉿eccc40e4b1fc)-[~/art]
└─# ./firstprog
Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!
```

Compilation vs Decompilation vs Disassemble

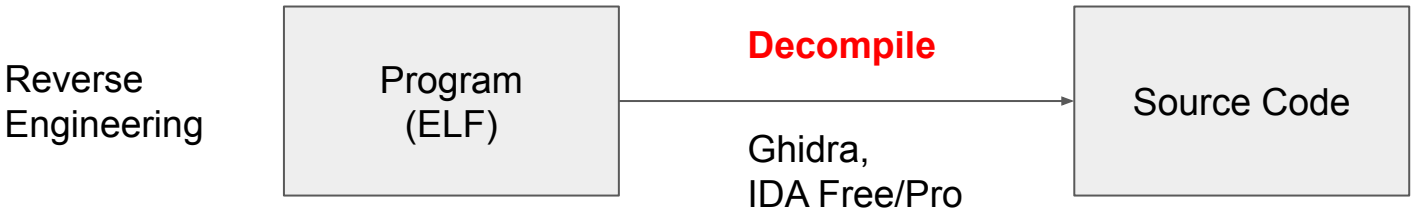
คำอธิบาย

Decompilation: การแปลง Machine Code กลับเป็นโค้ดที่มนุษย์อ่านเข้าใจได้ (ใกล้เคียงกับโค้ดต้นฉบับ)

IDA Free -> กดปุ่ม F5

IDA View-A | Pseudocode-A | Hex View-1 | Structures

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int String; // [esp+1Ch] [ebp-Ch]
4
5    puts("############################################################");
6    puts("##        Bienvennue dans ce challenge de cracking        ##");
7    puts("############################################################\n");
8    printf("Veuillez entrer le mot de passe : ");
9    String = getString();
10   if ( !strcmp(String, "123456789") )
11     printf("Bien joue, vous pouvez valider l'epreuve avec le pass : %s!\n", "123456789");
12   else
13     puts("Dommage, essaye encore une fois.");
14   return 0;
15 }
```



Decompilation - https://dogbolt.org/

Upload File

Your file must be **less than 2MB** in size. Uploaded binaries are retained.

Choose File | firstprog

Ghidra

10.3.2 (1e9ed15e)

```
157
158 undefined4 main(void)
159
160 {
161   int local_14;
162
163   for (local_14 = 0; local_14 < 10; local_14 = local_14 + 1) {
164     puts("Hello, world!\n");
165   }
166   return 0;
167 }
168
169
170
171 // WARNING: Function: __x86.get_pc_thunk.bx replaced with injectio
172
173 void _fini(void)
174
175 {
```

angr

9.2.60

```
166 int main()
167 {
168     char *v0;  // [bp-0x30]
169     void* v1;  // [bp-0x14]
170     unsigned int v2;  // [bp-0x10]
171     unsigned int v3;  // [bp-0x4]
172     unsigned int v4;  // [bp+0x0]
173
174     v3 = v4;
175     v2 = stack_base + 4;
176     for (v1 = 0; v1 <= 9; v1 += 1)
177     {
178         v0 = "Hello, world!\n";
179         puts(v0);
180     }
181     return;
182 }
183
184 int _fini()
185 {
```

Decompilation - https://dogbolt.org/

Upload File

Your file must be **less than 2MB** in size. Uploaded binaries are retained.

Choose File | firstprog

BinaryNinja

3.4.4271 (b7fd028d)

```
}

int32_t main(int32_t argc, char** argv, char** envp)
{
    void* const var_4 = __return_addr;
    int32_t* var_10 = &argc;
    for (int32_t var_14 = 0; var_14 <= 9; var_14 = (var_14 + 1))
    {
        puts("Hello, world!\n");
    }
    return 0;
}

int32_t _fini()
{
    return;
}
```

Hex-Rays

8.3.0.230608

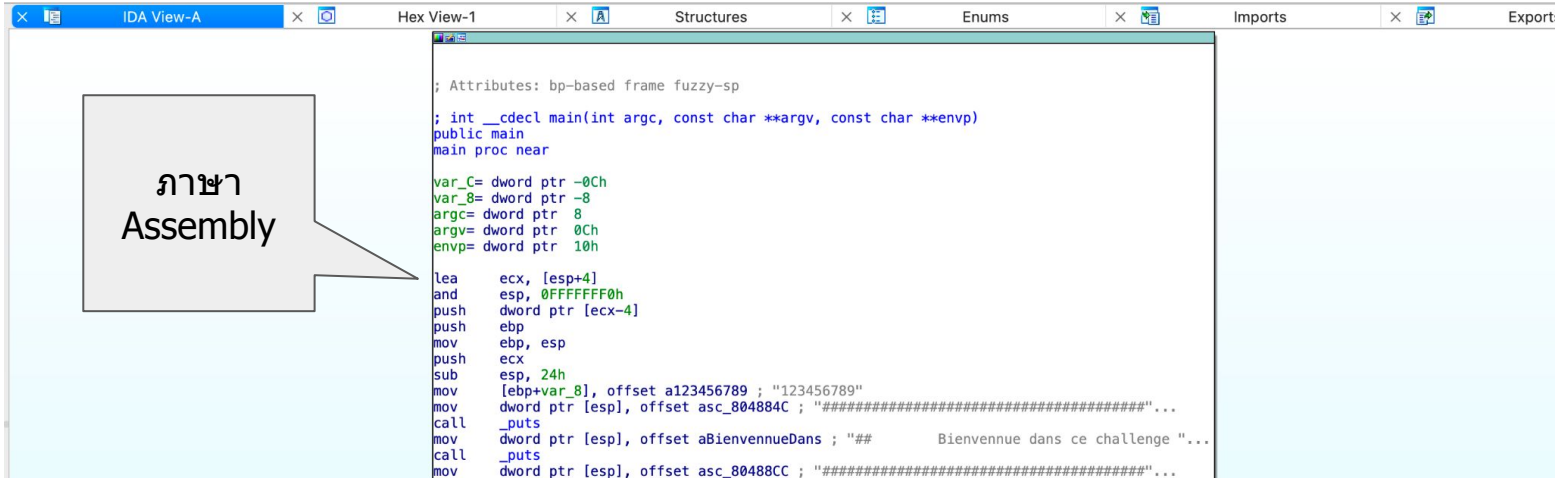
```
//----- (0000118D) --------------------------------------------------------
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // [esp+0h] [ebp-Ch]

  for ( i = 0; i <= 9; ++i )
    puts("Hello, world!\n");
  return 0;
}

//----- (000011E0) --------------------------------------------------------
void term_proc()
{
  ;
}

// nfuncs=18 queued=12 decompiled=12 lumina nreq=0 worse=0 better=
// ALL OK, 12 function(s) have been successfully decompiled
```

Compilation vs Decompilation vs Disassemble



Reverse Engineering

Program (ELF)

Disassemble

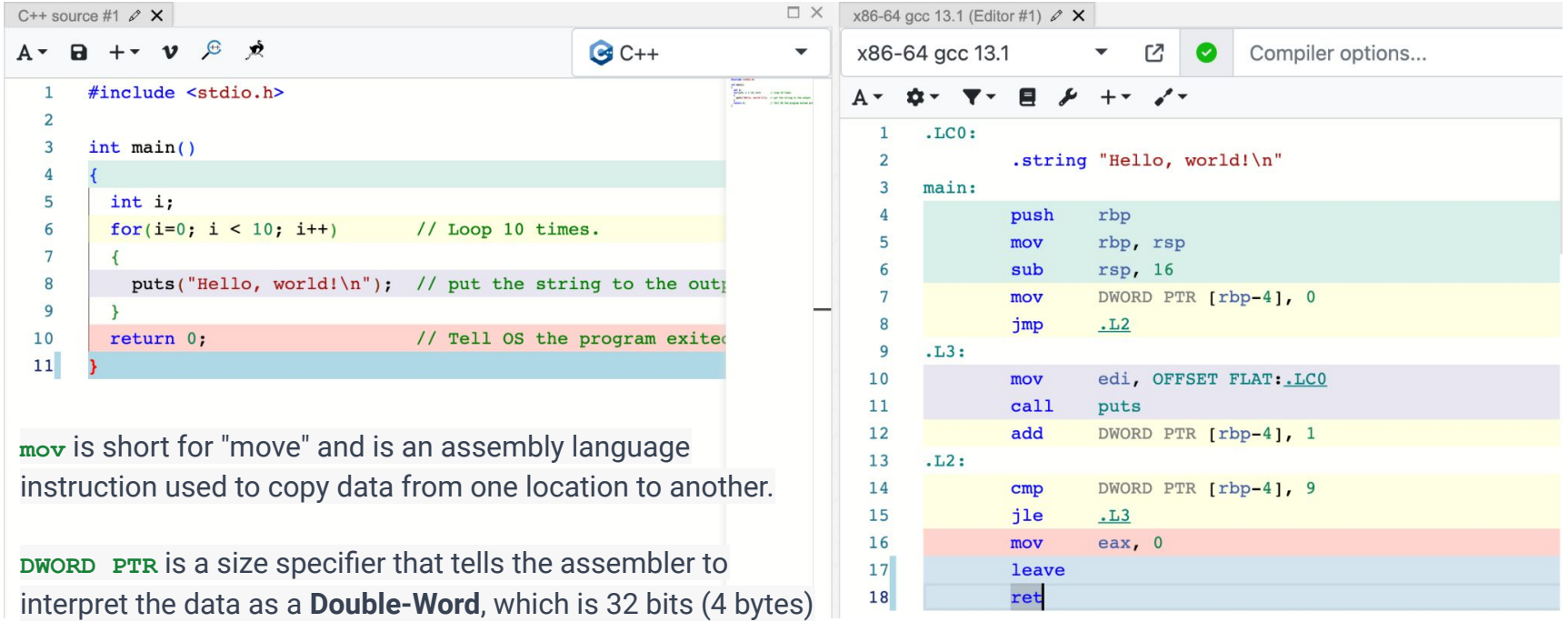
Ghidra, radare2, objdump, gdb, IDA Free/Pro

Assembly Code

คำอธิบาย

Disassemble: การแปลง **Machine Code** เป็น **Assembly Code** ซึ่งเป็นโค้ดระดับต่ำที่มนุษย์อ่านได้ แต่ซับซ้อนกว่าโค้ดต้นฉบับ

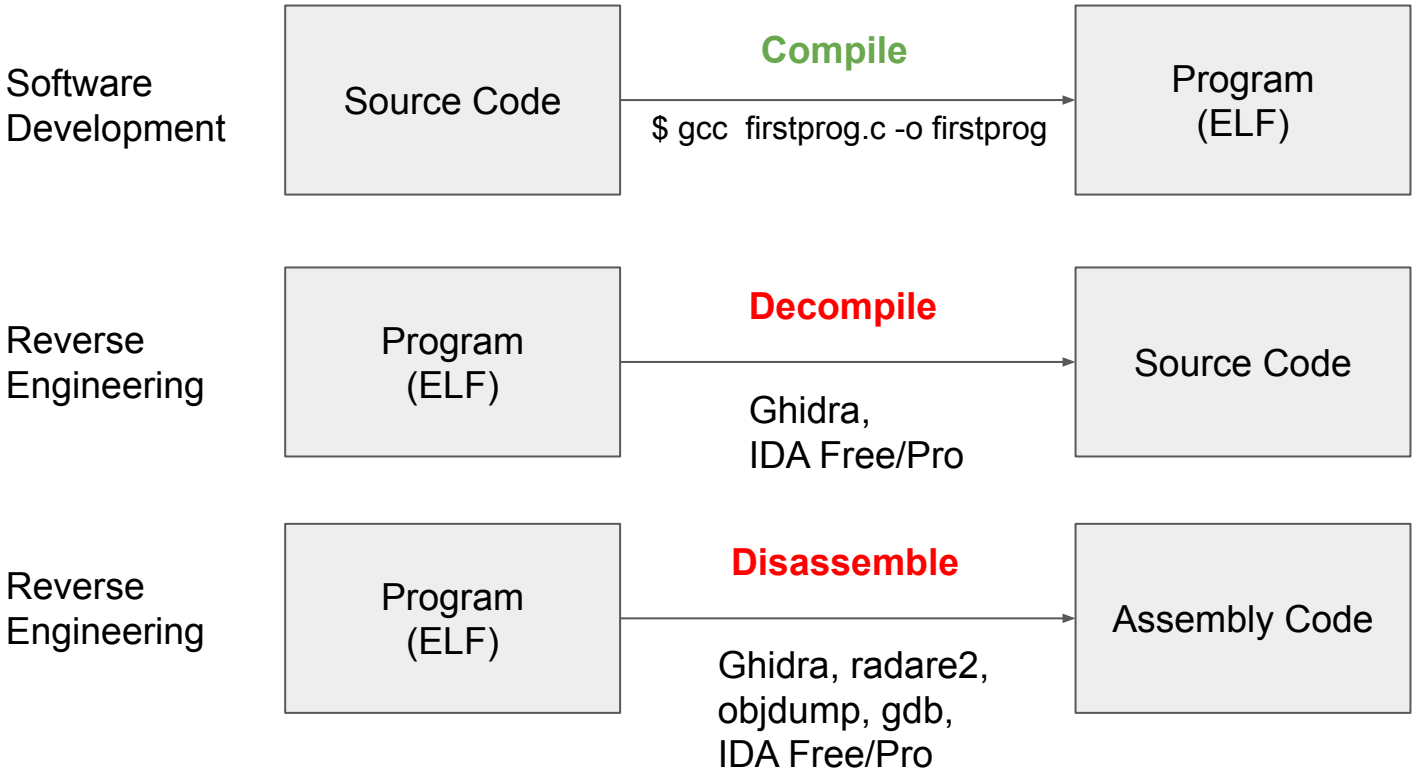
Disassemble - https://gcc.godbolt.org/



`mov` is short for "move" and is an assembly language instruction used to copy data from one location to another.

`DWORD PTR` is a size specifier that tells the assembler to interpret the data as a **Double-Word**, which is 32 bits (4 bytes) on most architectures.

Compilation vs Decompilation vs Disassemble



คำอธิบาย

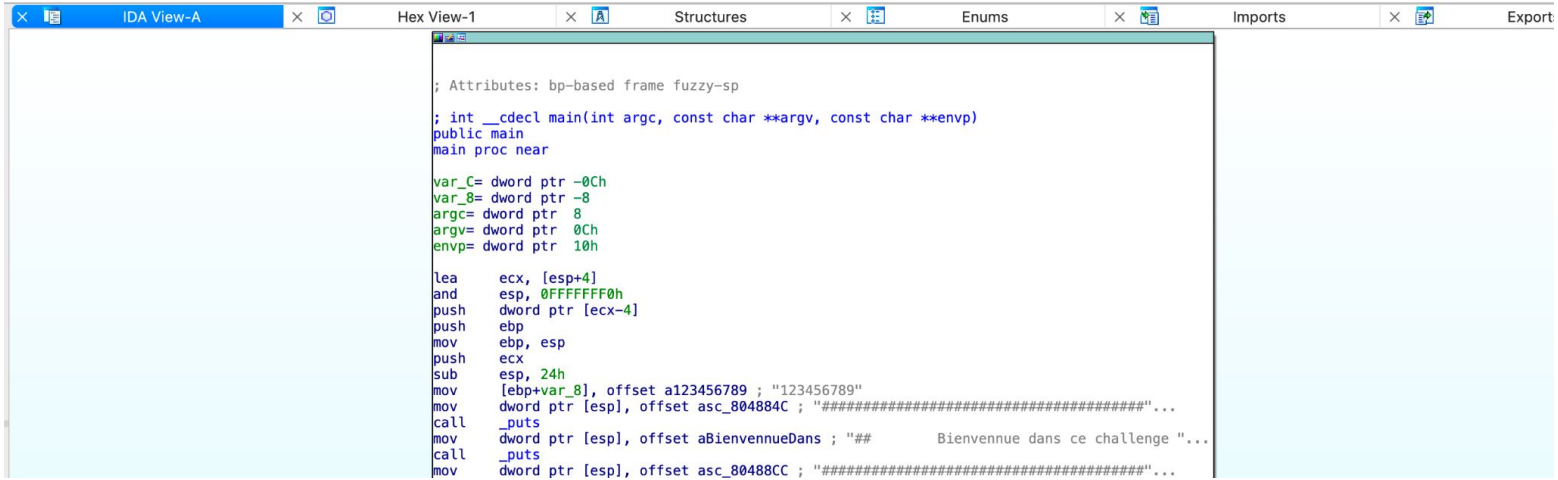
Compilation: แปลงโค้ดมนุษย์เป็น Machine Code

Decompilation: แปลง **Machine Code** กลับเป็นโค้ดมนุษย์

Disassemble: แปลง **Machine Code** เป็น **Assembly Code**

คำถามที่น่าสนใจ

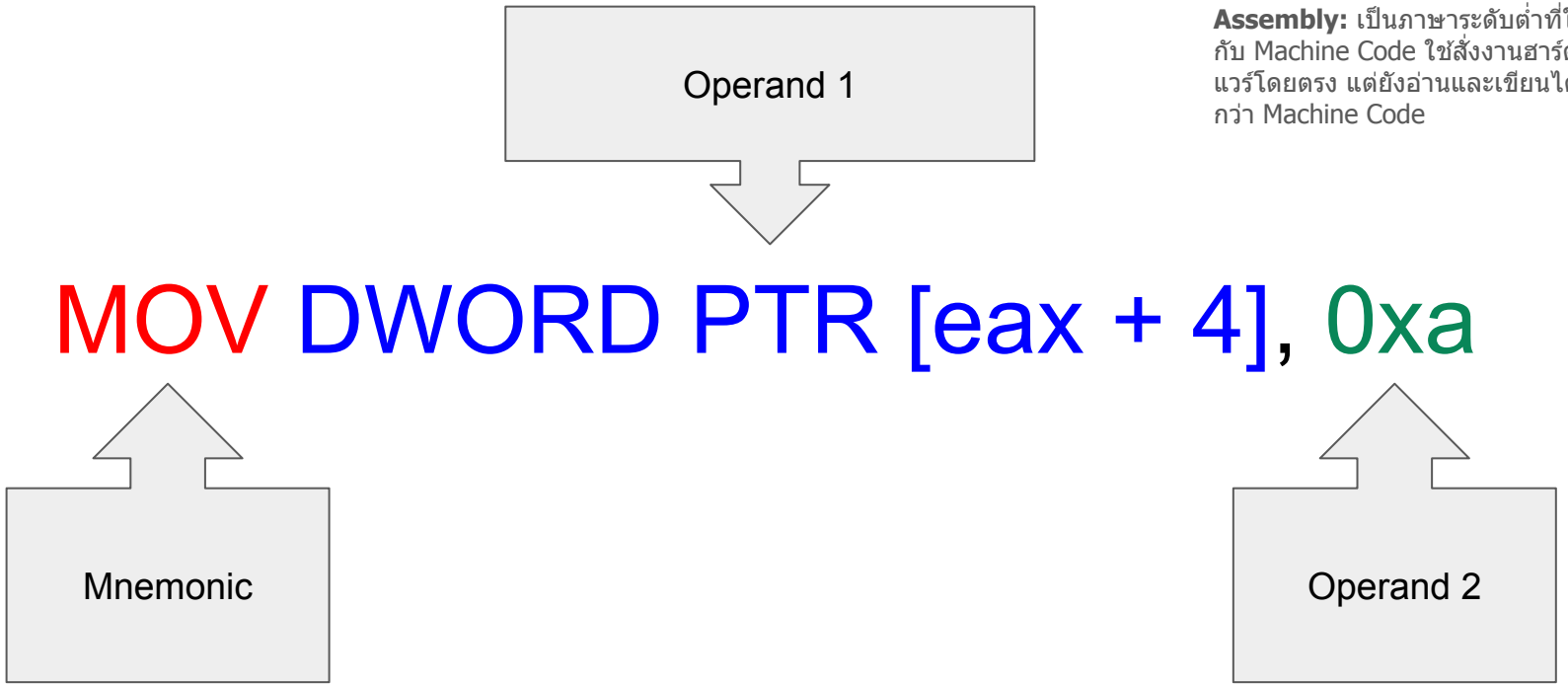
“ถ้าเราสามารถ Decompile ได้ ทำไมเรายังต้อง Disassemble?”



ภาษา Assembly 101

คำอธิบาย

Assembly: เป็นภาษาระดับต่ำที่ใกล้เคียงกับ Machine Code ใช้สั่งงานฮาร์ดแวร์โดยตรง แต่ยังอ่านและเขียนได้ง่ายกว่า Machine Code



ภาษา Assembly 101

c7 Opcode (run by the machine)

a line of instruction (before decode)

Bytecode (run in VM e.g. Java, CLR, Android Dalvik)

c7 40 04 0a 00 00 00

ภาษา Assembly 101

c7 Opcode (run by the machine)

a line of instruction (before decode)

a line of instruction (after 32-bit decode)

c7 40 04 0a 00 00 00

MOV DWORD PTR [eax + 4], 0xa

ภาษา Assembly 101

c7 Opcode (run by the machine)

a line of instruction (before decode)

a line of instruction (after 32-bit decode)

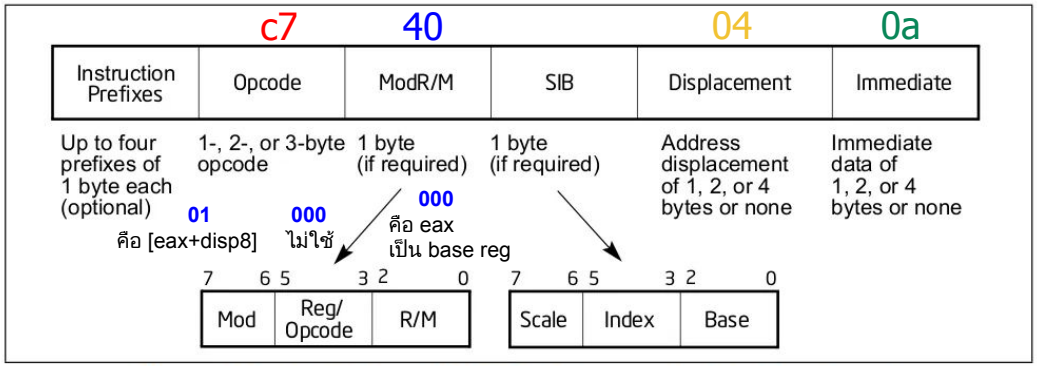
c7 40 04 0a 00 00 00

MOV DWORD PTR [eax + 4], 0xa

r8(r)			AL
r16(r)			AX
r32(r)			EAX
mm(r)			MM0
xmm(r)			XMM0
(In decimal) /digit (Opcode)			0
(In binary) REG =			000
Effective Address	**Mod**	**R/M**	
[EAX]	00	000	00
[ECX]		001	01
[EDX]		010	02
[EBX]		011	03
[--][--]		100	04
disp32[1]		101	05
[ESI]		110	06
[EDI]		111	07
[EAX]+disp8[2]	01	000	40
[ECX]+disp8		001	41
[EDX]+disp8		010	42
[EBX]+disp8		011	43
[--][--]+disp8		100	44
[EBP]+disp8		101	45
[ESI]+disp8		110	46
[EDI]+disp8		111	47

(with 32 bit immediate)

0100 0000 = 40
1000 0000 = 80



40 สำหรับ 8-bit displacement (mod: 01)
80 สำหรับ 32-bit displacement (mod: 10)
eax เป็น base reg คือ 000

Source: Intel® 64 and IA-32 Architectures Software Developer Manual Vol. 2

objdump: AT&T vs Intel Syntax

- objdump สามารถ Disassemble ไฟล์ Binary กลับเป็นโค้ด Assembly ได้
- $ objdump -Mintel -D ./<ชื่อไฟล์>

โค้ด Assembly
จะมี Syntax 2 แบบคือ
- AT&T Syntax
  - op <src>, <dest>
- Intel
  - op <dest>, <src>

```
$ objdump -Mintel -D ./hello
[...]
0000000000001145 <main>:
    1145:    55                      push   rbp
    1146:    48 89 e5                mov    rbp,rsp
    1149:    48 8d 3d b4 0e 00 00    lea    rdi,[rip+0xeb4]        # 2004 <_IO_stdin_used+0x4>
    1150:    e8 db fe ff ff          call   1030 <puts@plt>
    1155:    bf 00 00 00 00          mov    edi,0x0
    115a:    e8 e1 fe ff ff          call   1040 <exit@plt>
    115f:    90                      nop
```

Instruction (64-bit) - No Operand

Instruction: 1100 0011 (base-2, binary)
Instruction: 0xc3 (base-16, hexadecimal)
Instruction: ret
Mnemonic: ret
Opcode: c3

Binary	Hexadecimal	Binary	Hexadecimal
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

โปรแกรม objdump เอาไว้ Disassemble ได้

```
┌──(root㉿eccc40e4b1fc)-[~/art]
└─# objdump -D firstprog -Mintel |grep -A20 main.:
0000000000001139 <main>:
    1139:       55                      push   rbp
    113a:       48 89 e5                mov    rbp,rsp
    113d:       48 83 ec 10             sub    rsp,0x10
    1141:       c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0
    1148:       eb 13                   jmp    115d <main+0x24>
    114a:       48 8d 05 b3 0e 00 00    lea    rax,[rip+0xeb3]
    1151:       48 89 c7                mov    rdi,rax
    1154:       e8 d7 fe ff ff          call   1030 <puts@plt>
    1159:       83 45 fc 01             add    DWORD PTR [rbp-0x4],0x1
    115d:       83 7d fc 09             cmp    DWORD PTR [rbp-0x4],0x9
    1161:       7e e7                   jle    114a <main+0x11>
    1163:       b8 00 00 00 00          mov    eax,0x0
    1168:       c9                      leave
    1169:       c3                      ret
```

Instruction (64-bit) - 2 Operands

```
0x8345FC01

8    -> 1000
3    -> 0011
4    -> 0100
5    -> 0101
F    -> 1111
C    -> 1100
0    -> 0000
1    -> 0001
```

Instruction: **10000011**01000101111111000000001 (base-2, binary)
Instruction: 0x**83**45fc01 (base-16, hexadecimal)
Instruction: add DWORD PTR [rbp-0x4], 0x1
Mnemonic: add
Opcode: 83
Operand 1: DWORD PTR [rbp-0x4]
Operand 2: 0x1

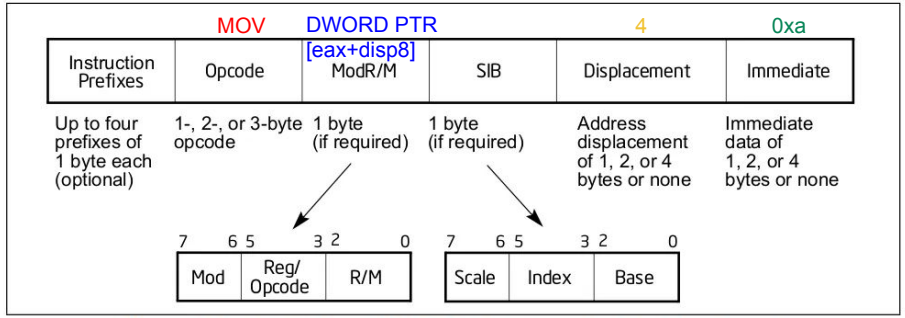
Instruction w/ no operand
- ret, nop, pushad, popad, mobsb

Instruction w/ 1 operand
- inc eax, dec ecx, push rdx, pop rcx

Instruction w/ 2 operands
- mov DWORD PTR [rbp-0x4], 0x0

```
┌──(root㉿eccc40e4b1fc)-[~/art]
└─# objdump -D firstprog -Mintel |grep -A20 main.:
0000000000001139 <main>:
    1139:       55                      push   rbp
    113a:       48 89 e5                mov    rbp,rsp
    113d:       48 83 ec 10             sub    rsp,0x10
    1141:       c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0
    1148:       eb 13                   jmp    115d <main+0x24>
    114a:       48 8d 05 b3 0e 00 00    lea    rax,[rip+0xeb3]
    1151:       48 89 c7                mov    rdi,rax
    1154:       e8 d7 fe ff ff          call   1030 <puts@plt>
    1159:       83 45 fc 01             add    DWORD PTR [rbp-0x4],0x1
    115d:       83 7d fc 09             cmp    DWORD PTR [rbp-0x4],0x9
    1161:       7e e7                   jle    114a <main+0x11>
    1163:       b8 00 00 00 00          mov    eax,0x0
    1168:       c9                      leave
    1169:       c3                      ret
```

สรุป ภาษา Assembly

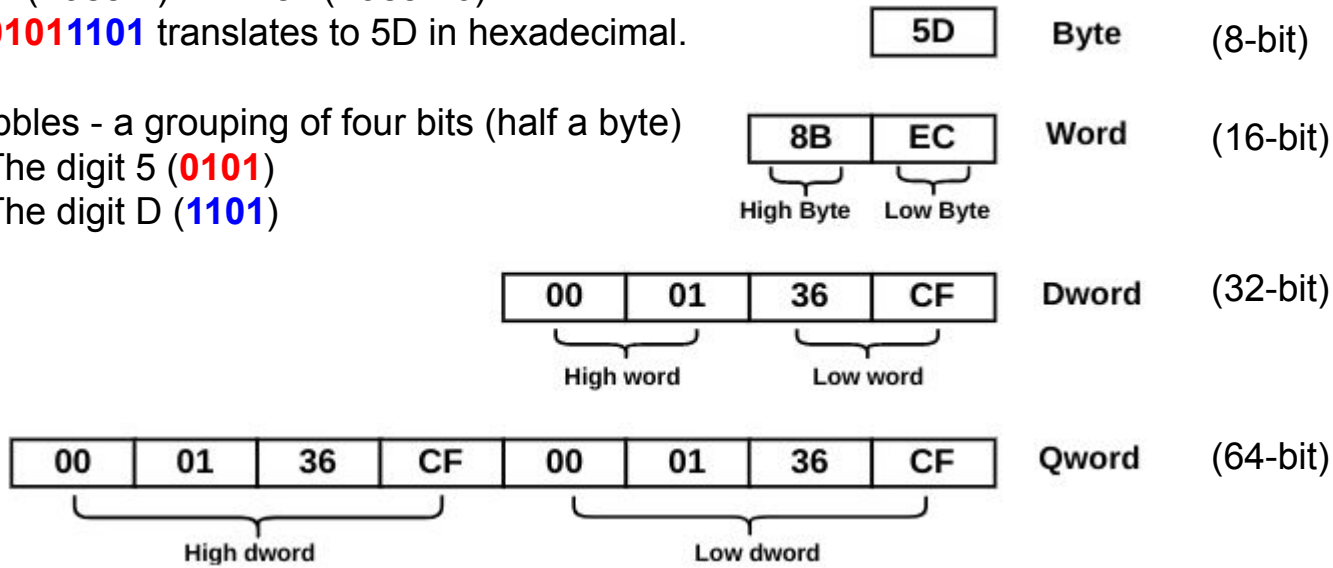


- Instruction Structure
  - Instruction c7 40 04 0a
  - Instruction (decoded) MOV DWORD PTR [eax + 4], 0xa
  - Opcode c7 (MOV) // เอาไว้ให้คอมอ่าน
  - Mnemonic MOV (move data) // เอาไว้ให้คนอ่าน
  - Operand 1 "*DWORD PTR [eax + 4]*"
    - Prefix DWORD PTR (คือ size ขนาด 4-byte) // ไม่มีในก่อน Decode
    - Mod R/M 40 (คือ eax เป็น base reg + 8-bit displacement)
    - Displacement 4 (จากใน [eax + 4])
  - Operand 2 "*0xa*"
    - Intermediate 0xa (คือค่าคงที่ Decimal 10)
- Address เป็น Operand ได้ เช่น 0x1234
- Architecture x86, x86_64, ARM *(** ติดไว้ก่อน)*
- Assembly Syntax Intex, AT&T
- Branching
- Calling Convention

Data Type

Bin (Base-2) <-> Hex (Base-16)
01011101 translates to 5D in hexadecimal.

nibbles - a grouping of four bits (half a byte)
The digit 5 (**0101**)
The digit D (**1101**)

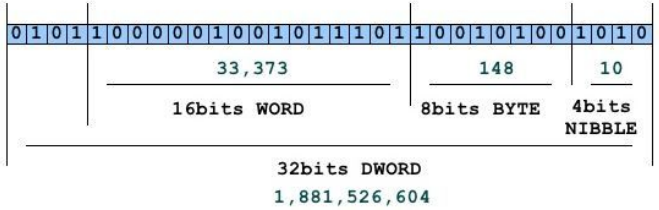


https://subscription.packtpub.com/book/security/9781788392501/4/ch04lvl1sec32/1-computer-basics

Data Type

Data Representations

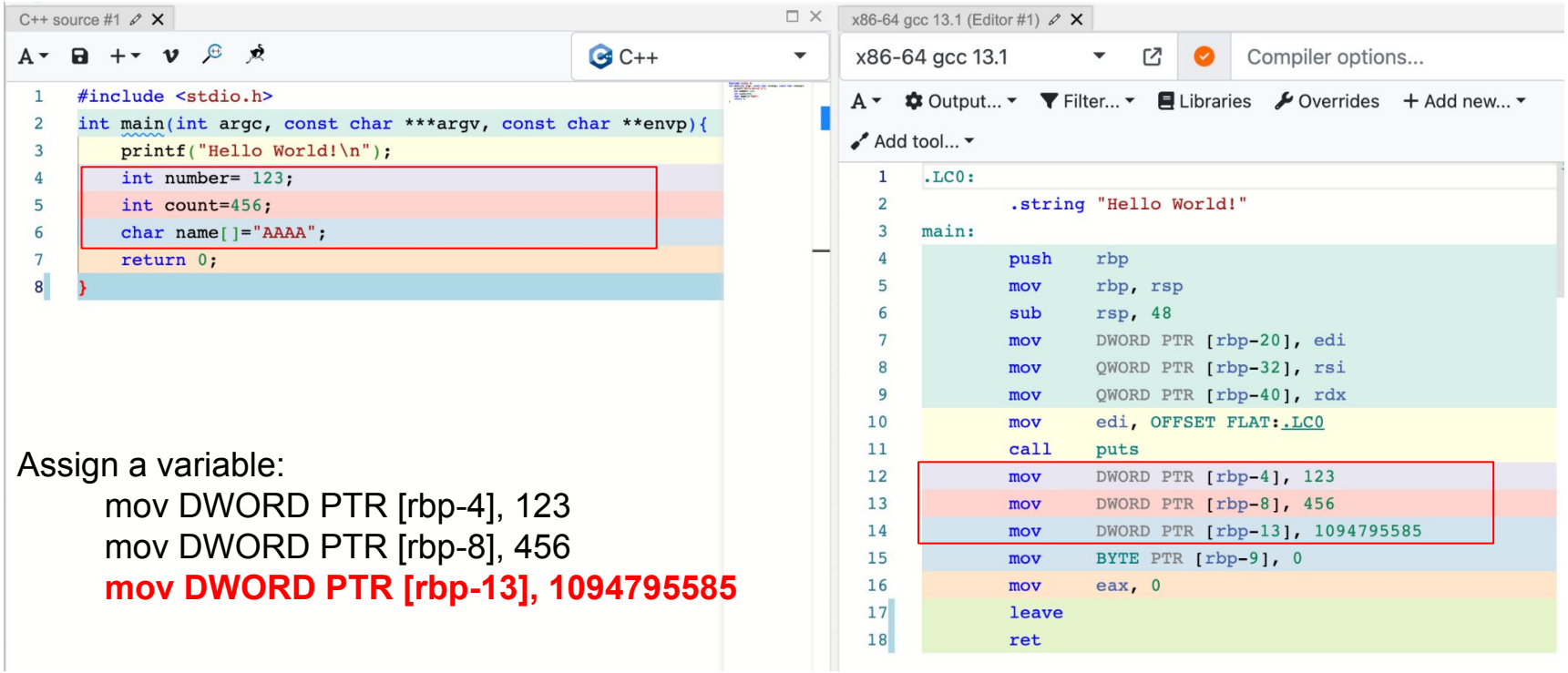
Bit:	1 bit (0/1)
Nibble:	4 bits (0-15)
Byte:	8 bits (0-255)
Word:	16 bits (0-65535)
Double Word(DWORD):	32 bits (0-4294967295)
Quad Word(QWORD):	64 bits (0-18446744073709551615)



https://learningnetwork.cisco.com/s/question/0D53i00000Kt7jJCAR/what-is-word-in-networking

Assign a Variable in x86_64 Assembly

https://gcc.godbolt.org



Assign a variable:

mov DWORD PTR [rbp-4], 123

mov DWORD PTR [rbp-8], 456

mov DWORD PTR [rbp-13], 1094795585

Assign a Variable in x86_64 Assembly

C++ source #1

```cpp
#include <stdio.h>
int main(int argc, const char ***argv, const char **envp){
    printf("Hello World!\n");
    int number= 123;
    int count=456;
    char name[]="AAAA";
    return 0;
}
```

To put it in simpler terms, this line of code is storing the value

- **1094795585** (decimal) or
- 0x**41**414141 (hexadecimal) or
- **A**AAA (ASCII Hex)

into a 32-bit memory location located 13 bytes below the base pointer (rbp) in the current function's stack frame.

Assign a variable:

mov DWORD PTR [rbp-4], 123
mov DWORD PTR [rbp-8], 456
mov DWORD PTR [rbp-13], 1094795585

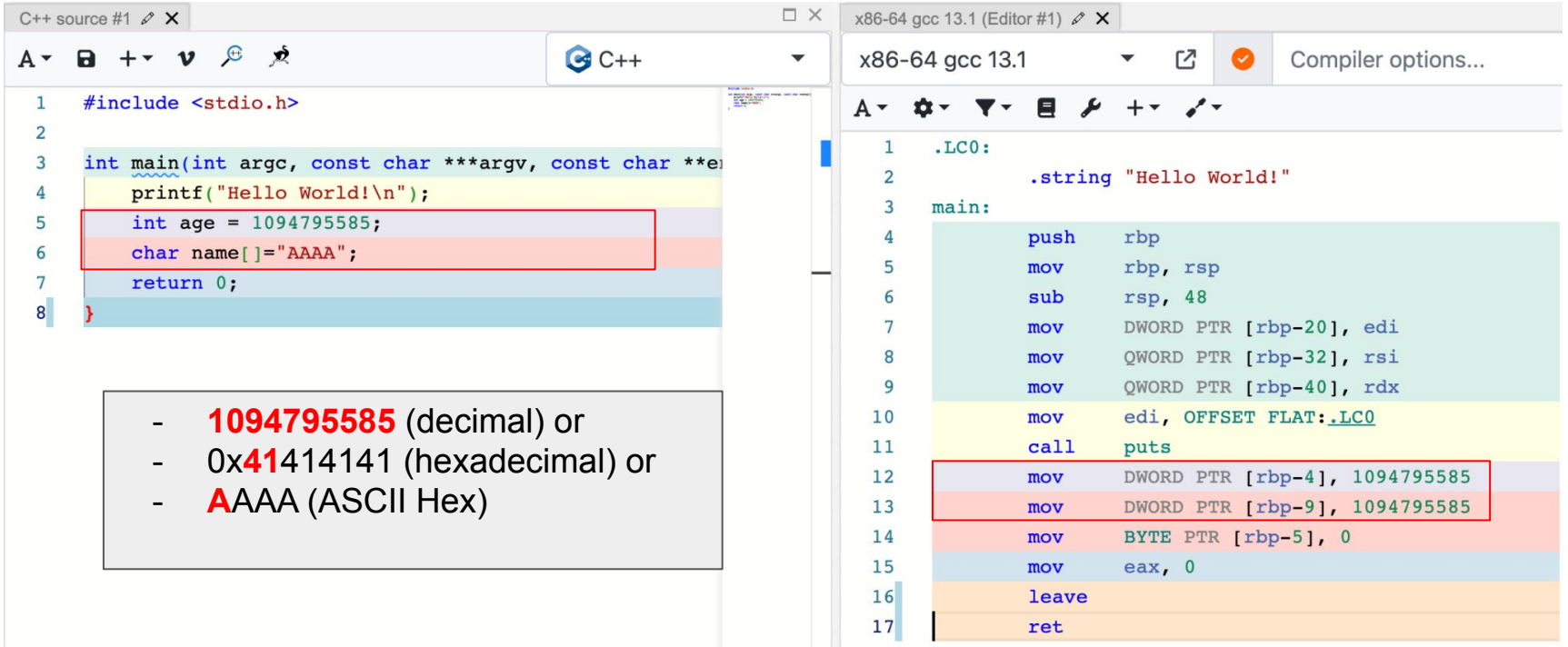
```
12      mov     DWORD PTR [rbp-4], 123
13      mov     DWORD PTR [rbp-8], 456
14      mov     DWORD PTR [rbp-13], 1094795585
15      mov     BYTE PTR [rbp-9], 0
16      mov     eax, 0
17      leave
18      ret
```

ASCII Table

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

https://simple.wikipedia.org/wiki/ASCII

Base Representation



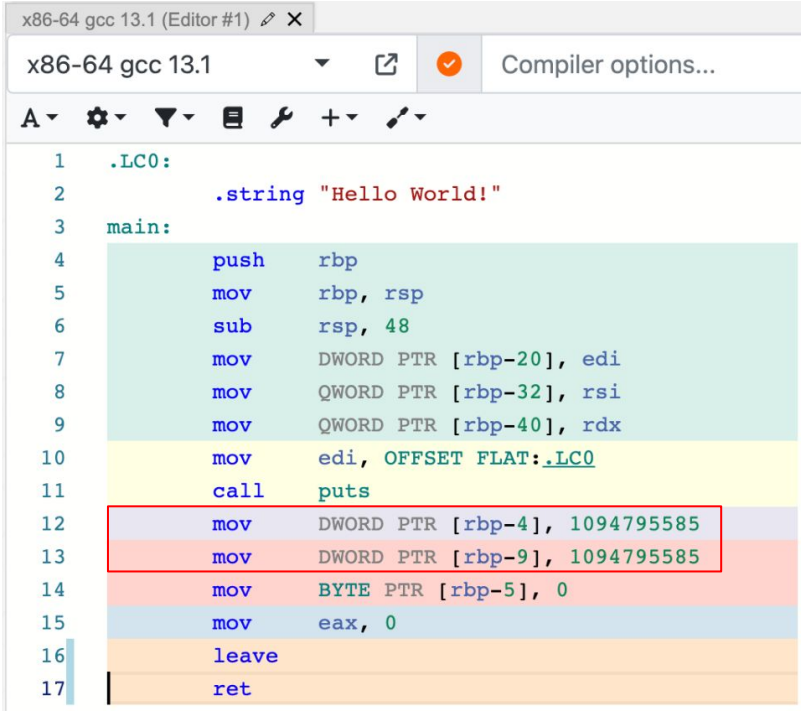
Question: Char Array or Int?

When we look at an assembly instruction, how we can determine if the “1094795585” in

mov DWORD PTR [rbp-13], 1094795585

is “AAAA” (char array) or “1094795585” (int) ?

```
int age = 1094795585;
char name[]="AAAA";
```



Answer: Char Array or Int?

1. If the surrounding code treats the value as a character array, then "1094795585" is likely to be interpreted as "AAAA".

For example:

assembly — Copy code

```
section .data
    my_array db 4 dup(0)  ; Allocate 4 bytes for the character array

section .text
    mov DWORD PTR [rbp-13], 1094795585  ; Store the value "1094795585" into
```

1. On the other hand, if the surrounding code treats the value as an integer, then "1094795585" would be interpreted as an integer.

For example:

assembly — Copy code

```
section .text
    mov eax, 1094795585  ; Move the value "1094795585" into the EAX register
    mov DWORD PTR [rbp-13], eax  ; Store the integer value in the memory loc
```

In summary, the determination of whether "1094795585" is a character array or an integer depends on the usage and context in the rest of the code. Assembly language itself does not provide explicit type information for constants, so the programmer must follow conventions and maintain consistency in how they interpret and use data in their program.

x86-64 gcc 13.1 (Editor #1)

x86-64 gcc 13.1 — Compiler options...

```
1   .LC0:
2           .string "Hello World!"
3   main:
4           push    rbp
5           mov     rbp, rsp
6           sub     rsp, 48
7           mov     DWORD PTR [rbp-20], edi
8           mov     QWORD PTR [rbp-32], rsi
9           mov     QWORD PTR [rbp-40], rdx
10          mov     edi, OFFSET FLAT:.LC0
11          call    puts
12          mov     DWORD PTR [rbp-4], 1094795585
13          mov     DWORD PTR [rbp-9], 1094795585
14          mov     BYTE PTR [rbp-5], 0
15          mov     eax, 0
16          leave
17          ret
```

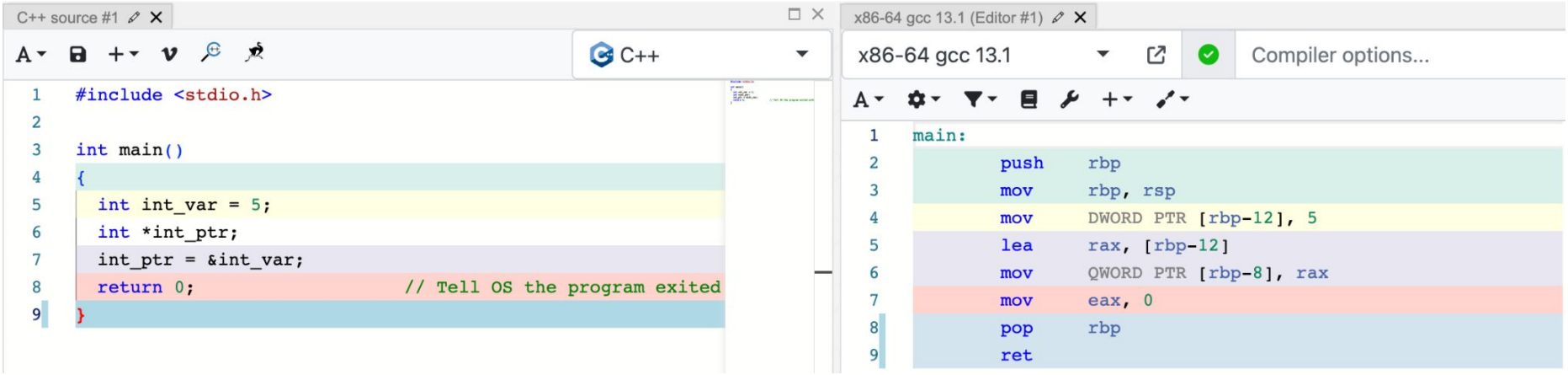
Pointer + "Load Effective Address" (LEA) operation

Pointer:

lea rax, [rbp-12]

mov QWORD PTR [rbp-8], rax

It calculates the memory address of [rbp-12] (the value stored at rbp - 12) and loads that address into the rax register.



The LEA instruction does not access the memory but merely calculates the address.

System Exploitation (Pwn)

CPU Architecture: x86 (32-bit), x64 (64-bit)

Register (ที่เก็บข้อมูลชั่วคราว)

- Instruction Pointer
  - EIP/RIP
- General Purpose
  - EAX/RAX, ECX/RCX, EDX/RDX, EBX/RBX

  (Accumulator, Counter, Data, and Base)
- Pointer and Index
  - ESI/RSI, EDI/RDI

  (Source/Dest Index)
  - ESP/RSP, EBP/RBP

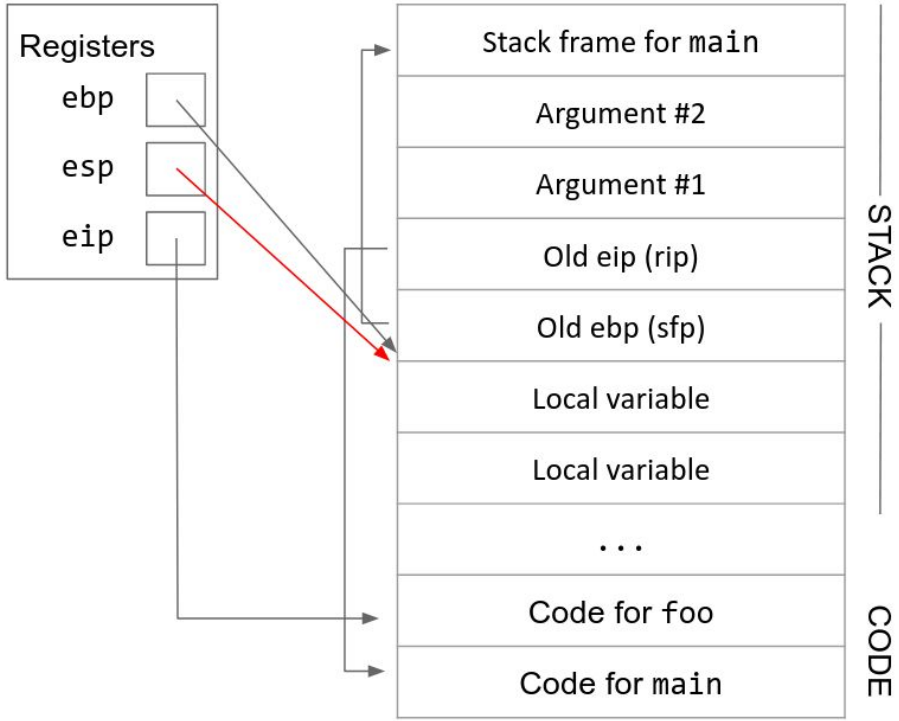
  (Stack Pointer, Base Pointer)
- Special
  - Rflags, eflags, flags
    - Carry flag (CF)

x64	x86		
8 bytes	4 bytes	2 bytes	1 byte
rax	eax	ax	al , ah
rcx	ecx	cx	cl , ch
rdx	edx	dx	dl , dh
rbx	ebx	bx	bl , bh
rsp	esp	sp	spl*
rbp	ebp	bp	bpl*
rsi	esi	si	sil*
rdi	edi	di	dil*
r8-r15	r8d-r15d*	r8w-r15w*	r8b-r15b*

CPU Architecture: x86 (32-bit), x64 (64-bit)

Register (ที่เก็บข้อมูลชั่วคราว)

- Instruction Pointer
  - EIP/RIP



https://textbook.cs161.org/memory-safety/x86.html

CPU Architecture: x86 (32-bit), x64 (64-bit)

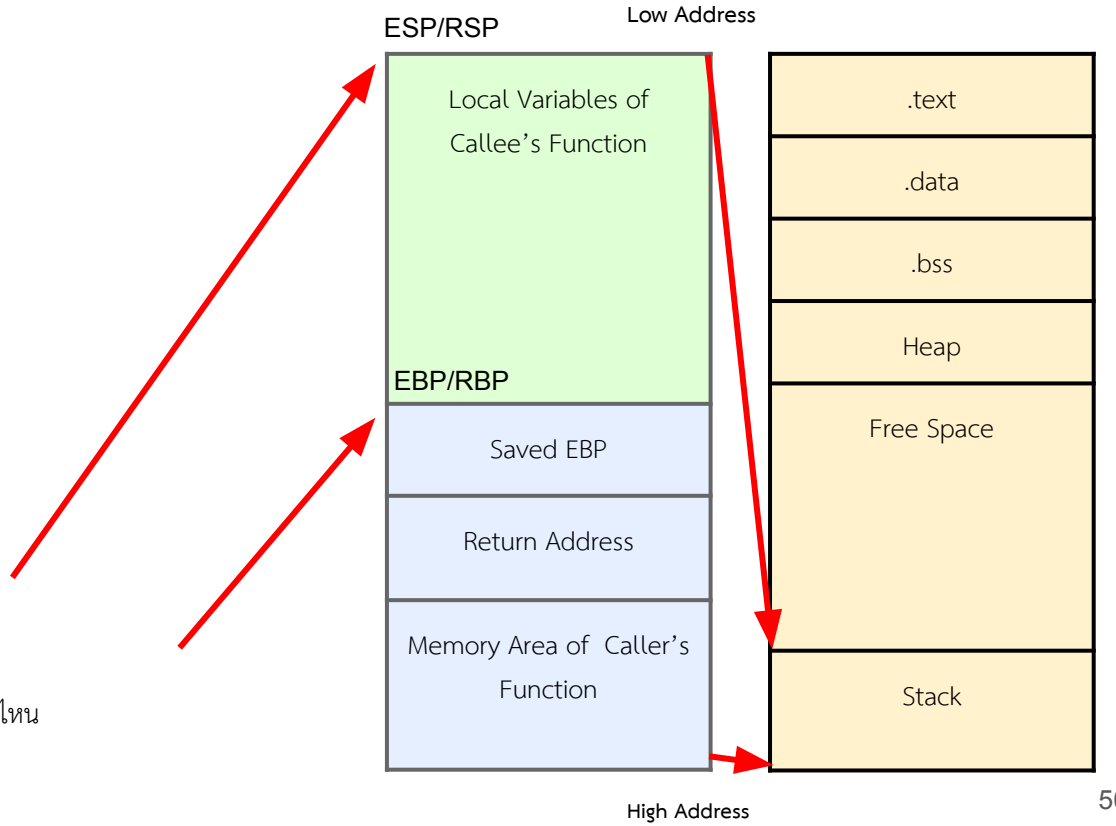
Assembly Instruction (คำสั่ง)

- **push** => เพิ่มค่าจาก Operand เข้าไปใน Stack
- **pop** => เอาค่าออกมาจาก Stack ไปเก็บใน Operand
- **call** => เรียก Function
- **ret** => Return กลับไปยัง Caller Function
  - **leave** => จัดการ Stack

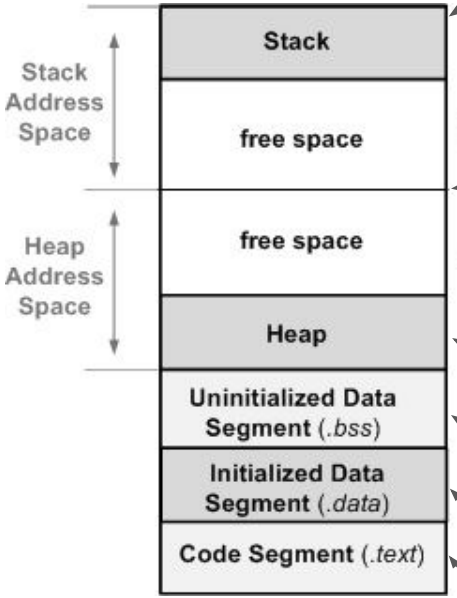
    // mov esp,ebp; pop ebp
- **nop** => No Operation หรือไม่ทำอะไรเลย

Register (ที่เก็บข้อมูลชั่วคราว)

- **ESP/RSP** => Stack Pointer หรือบขอบบนของ Stack
- **EBP/RBP** => Base Pointer หรือขอบล่างของ Stack
- **EIP/RIP** => Instruction Pointer ชี้ว่าคำสั่งต่อไป Execute ที่ไหน



Memory Layout



Stack: ใช้เก็บข้อมูลการเรียกฟังก์ชันและ **Local Variable** เติบโตจากที่อยู่หน่วยความจำสูงไปต่ำ

Free Space: พื้นที่ว่างระหว่าง **Stack** และ **Heap**

Heap: ใช้สำหรับการจัดสรรหน่วยความจำแบบ **Dynamic** เช่น ข้อมูลที่สร้างขึ้นขณะรันโปรแกรม เติบโตจากที่อยู่ต่ำไปสูง

BSS Segment: เก็บตัวแปร **global** และ **static** ที่ยังไม่ได้กำหนดค่า

Data Segment: เก็บตัวแปร **global** และ **static** ที่ถูกกำหนดค่าแล้ว

Text Segment: เก็บคำสั่งโปรแกรม (โค้ด)

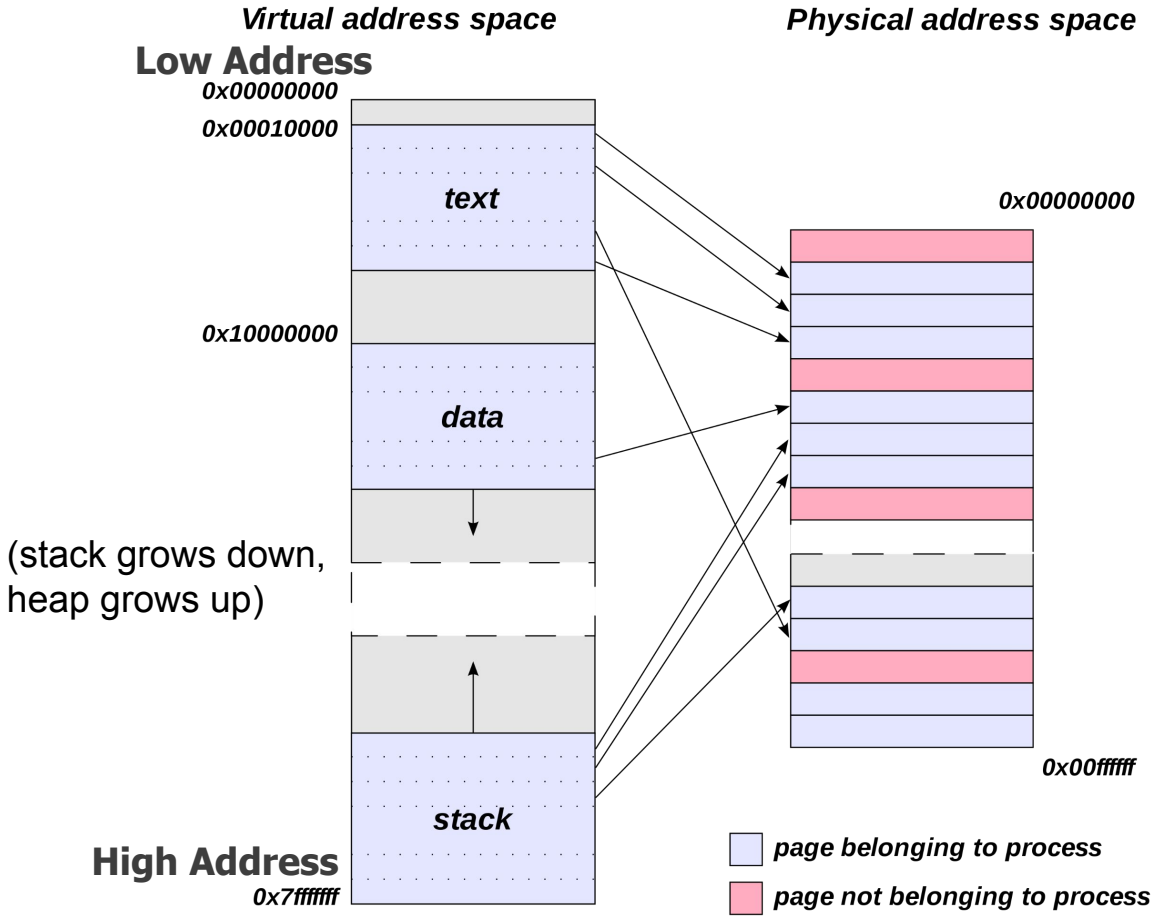
คำอธิบายการเติบโต (Growth):

- **Heap** เติบโตจากที่อยู่หน่วยความจำ**ต่ำไปสูง (ขึ้น)**
- **Stack** เติบโตจากที่อยู่หน่วยความจำ**สูงไปต่ำ (ลง)**

Virtual Address Space

 # High Address at the Bottom
gdb
IDA Pro
(Paul Chin)

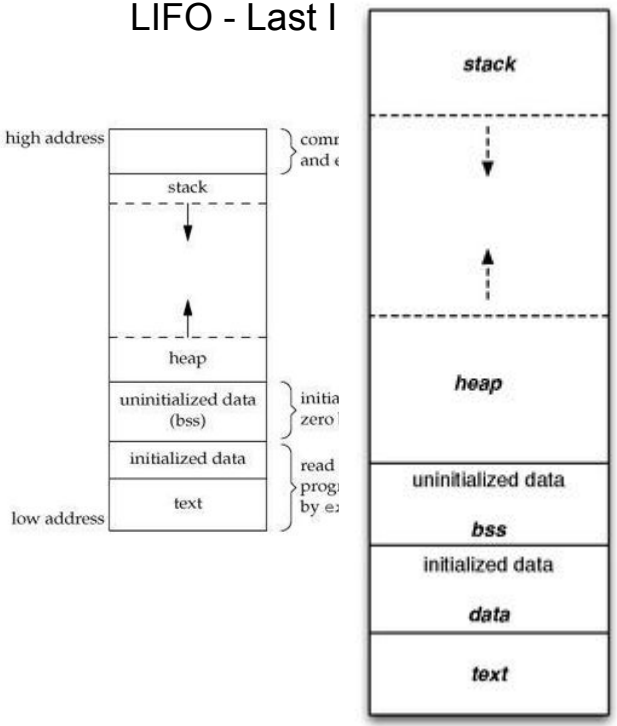
 # High Address at the Top
Ghidra
Radare2

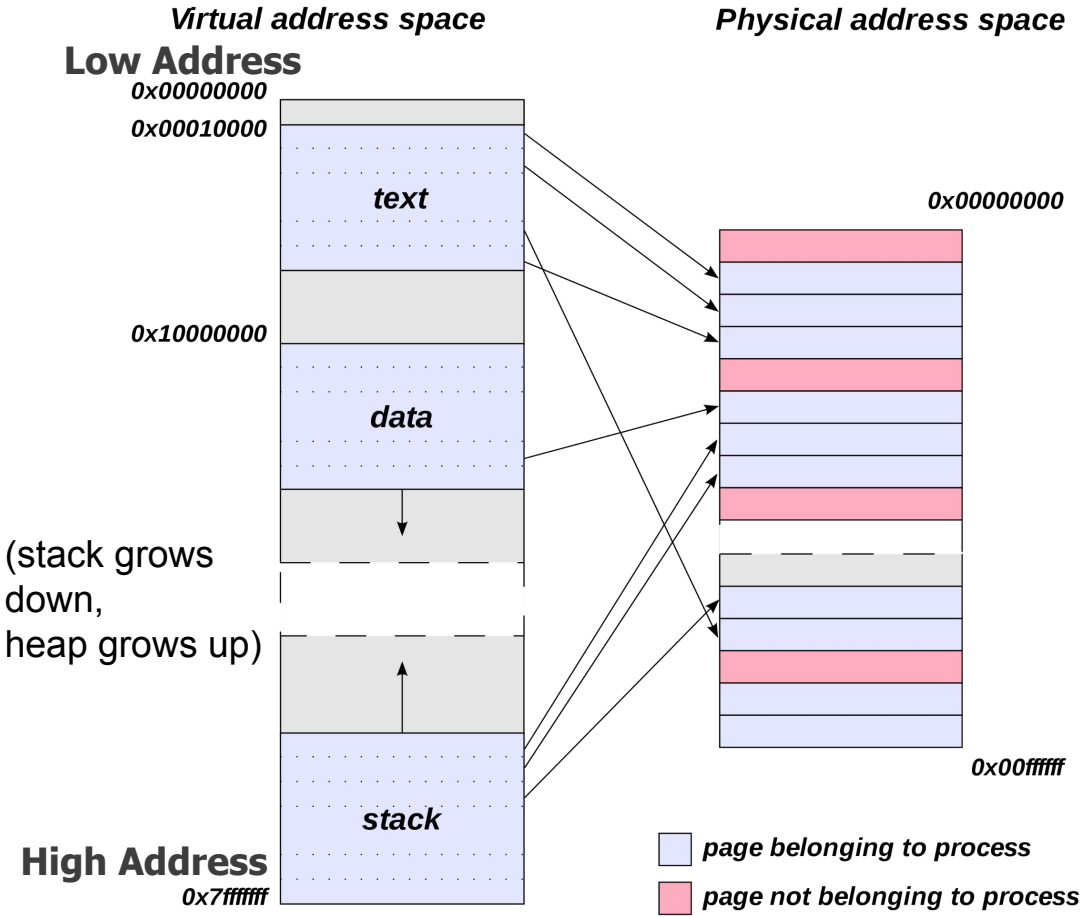


https://en.wikipedia.org/wiki/Address_space

Stack (Linux - glibc)

LIFO - Last I





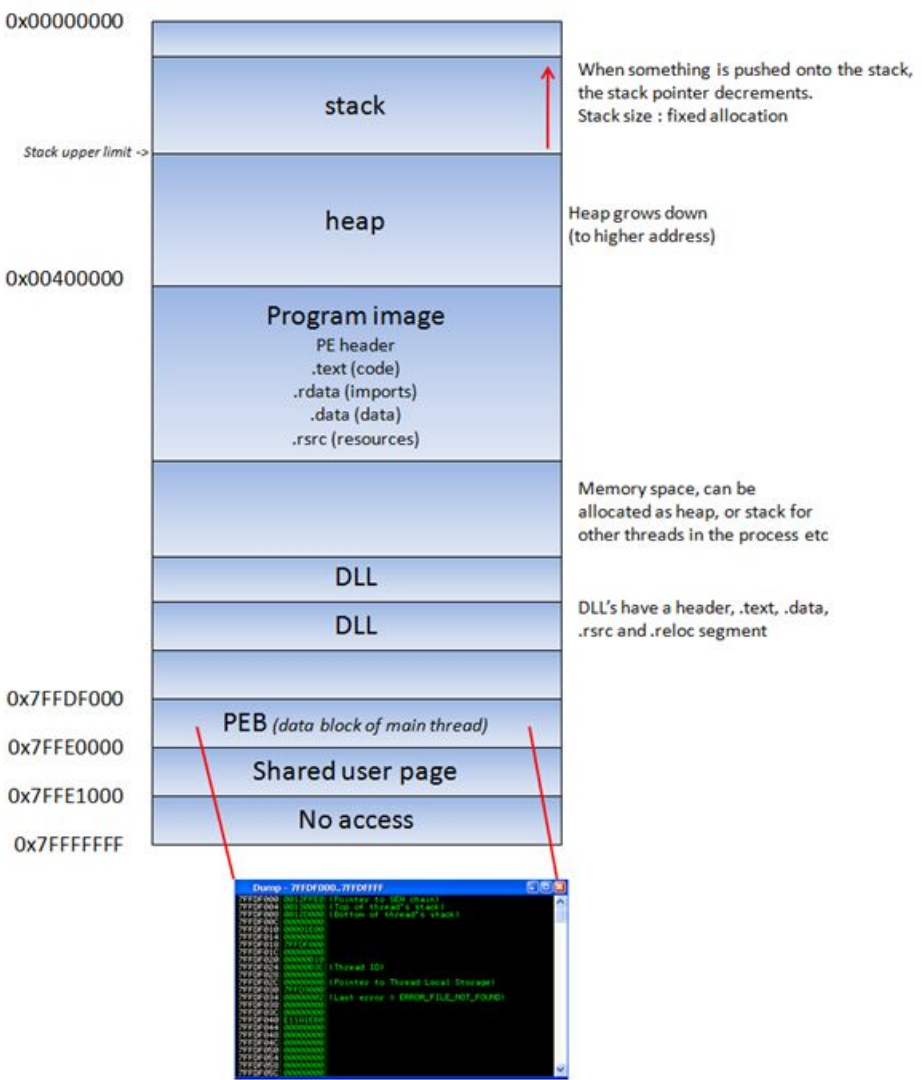
Stack (Windows - CRT)

Low Address

LIFO - Last In First Out

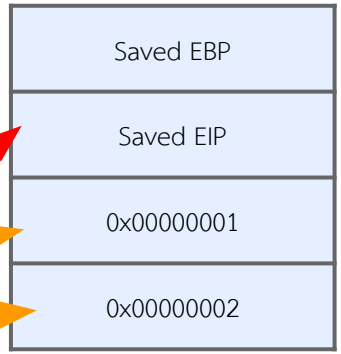
Thread Stack Frame Example	
Function A return address:	0x00401024
Parameter 1 for function A:	0x00000040
Parameter 2 for function A:	0x00001000
Parameter 3 for function A:	0xFFFFFFFF

High Address



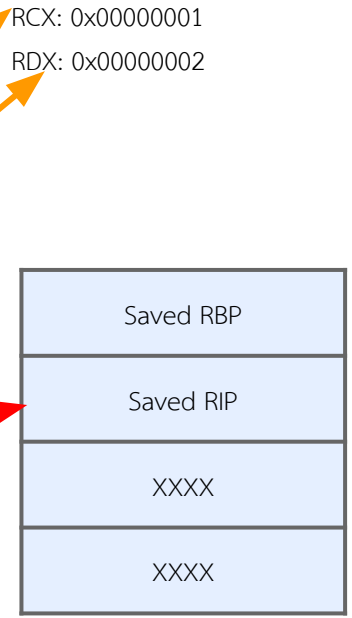
x86 Calling Convention

```
1 int sum(int a, int b){
2     return a+b;
3 }
4
5 void main(){
6     int x = sum(1,2);
7     printf("%d", x);
8 }
```



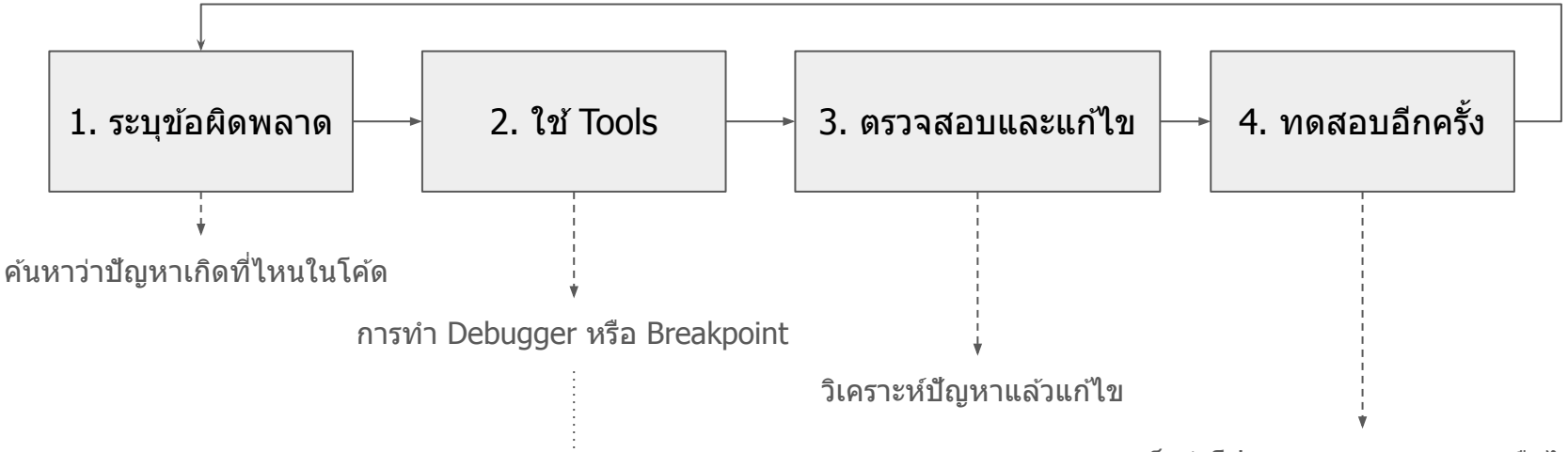
x64 Calling Convention

```
1 int sum(int a, int b){
2     return a+b;
3 }
4
5 void main(){
6     int x = sum(1,2);
7     printf("%d", x);
8 }
```



Debugging

ขั้นตอนการดีบัก:



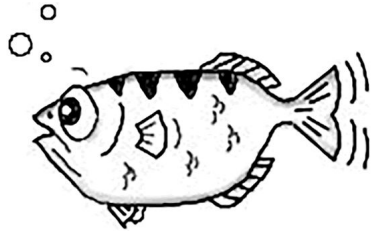
เครื่องมือ
Debugger: ตรวจสอบโปรแกรมขณะรัน
Breakpoint: หยุดโค้ดชั่วคราว
Logging: บันทึกผลการทำงานของโปรแกรม

คำอธิบาย

Debugging: กระบวนการค้นหาและแก้ไขข้อผิดพลาด (บั๊ก) ในโปรแกรม เพื่อให้ทำงานได้ถูกต้อง

gdb

```
*RAX  0x1c
 RBX  0x0
*RCX  0x7fffffffeca8 —▸ 0x7fffffffeeaf ◂— 0x4f494e4f48545950 ('PYTHONIO')
*RDX  0x7ffff7de8a50 (_dl_fini) ◂— push   rbp
*RDI  0x7ffff7ffe168 ◂— 0x0
*RSI  0x1
*R8   0x7ffff7ffe6f8 ◂— 0x0
 R9   0x0
 R10  0x0
*R11  0x1
*R12  0x4006b0 ◂— xor    ebp, ebp
*R13  0x7fffffffec90 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x0
 RSP  0x7fffffffec90 ◂— 0x1
*RIP  0x4006b0 ◂— xor    ebp, ebp
[ DISASM ]
 ► 0x4006b0    xor    ebp, ebp
   0x4006b2    mov    r9, rdx
   0x4006b5    pop    rsi
   0x4006b6    mov    rdx, rsp
   0x4006b9    and    rsp, 0xfffffffffffffff0
   0x4006bd    push   rax
   0x4006be    push   rsp
   0x4006bf    mov    r8, 0x4009c0
   0x4006c6    mov    rcx, 0x400950
   0x4006cd    mov    rdi, 0x4007a6
   0x4006d4    call   0x400680
[ STACK ]
00:0000| r13 rsp  0x7fffffffec90 ◂— 0x1
01:0008|          0x7fffffffec98 —▸ 0x7fffffffee7b ◂— 0x2f6465726168532f ('/Shared/')
02:0010|          0x7fffffffeca0 ◂— 0x0
03:0018| rcx      0x7fffffffeca8 —▸ 0x7fffffffeeaf ◂— 0x4f494e4f48545950 ('PYTHONIO')
04:0020|          0x7fffffffecb0 —▸ 0x7fffffffeec6 ◂— 0x79786f72705f6f6e ('no_proxy')
05:0028|          0x7fffffffecb8 —▸ 0x7fffffffeee3 ◂— 0x454d414e54534f48 ('HOSTNAME')
06:0030|          0x7fffffffecc0 —▸ 0x7fffffffeef9 ◂— 0x313d4c564c4853 /* 'SHLVL=1' */
07:0038|          0x7fffffffecc8 —▸ 0x7fffffffef01 ◂— 'HOME=/root'
[ BACKTRACE ]
 ► f 0          4006b0
   f 1               1
   f 2     7fffffffee7b
   f 3               0
Breakpoint *0x4006b0
pwndbg>
```

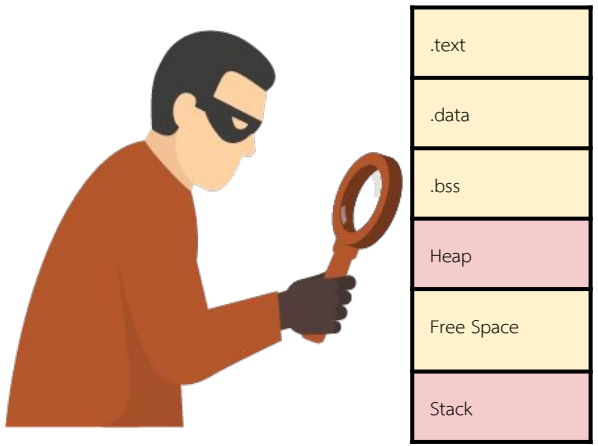


GDB
The GNU Project Debugger

Memory Corruption

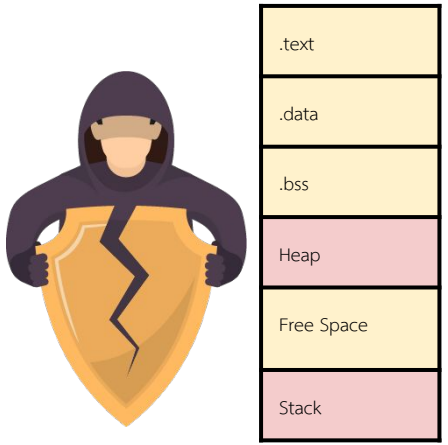
Memory Arbitrary Read

สามารถไปอ่านค่า Memory ในตำแหน่งที่ไม่สมควรได้



Memory Arbitrary Write

สามารถไปเขียนค่า Memory ในตำแหน่งที่ไม่สมควรได้



How to do that ?

Corrupt Program Logic

Today Topics:

- Buffer Overflow
- Format String
- Integer Overflow
- Array Index out of Bounds

Others:

- Off by one
- Use After Free (UAF)
- Double Free
- ...

Google » Chrome : Security Vulnerabilities (Execute Code)

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9
Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits D

Total number of vulnerabilities : **121** Page : 1 (This Page) 2 3

Copy Results Download Results

#	CVE ID	CWE ID	# of Exp its	Vulnerabi ty Type(s)	Publish Date	Update
1	CVE-2019-5819	20		E	2019-06-27	2019-0
Insufficient data validation in developer ols m n OS X prior to 74.0.3729.108 allowed a						
2	CVE-2019-5790	190		rflow	2019-05-23	2019-0
An integer overflow leading to an incorre capa in JavaScript in Google Chrome prior to 73.						
3	CVE-2019-5789	190		Exec Code Overf w	2019-05-23	2019-0
An integer overflow that leads to a use-after-free in WebMIDI in Google Chrome on Windo s prior to 73.0. crafted HTML page.						
4	CVE-2019-5788	190		Exec Code Overflow	2019 23	2019-0
An integer overflow that leads to a use-after-free in Blink Storage in Google Chrome on Linux prior to 73.0 a crafted HTML page.						
5	CVE-2019-5782	20		Exec Code	2019-02-19	2019-0
Incorrect optimization assumptions in V8 in Google Chrome p .3626.81 allowed a remote attack						
6	CVE-2019-5774	20		Exec Code	2019-02-19	2019-0
Omission of the .desktop filetype from the Safe Browsing chec feBrowsing in Google Chrome on L arbitrary code via a downloaded .desktop file.						
7	CVE-2019-5771	119		Exec Code Overflow	2019-02-19	2019-0
An incorrect JIT of GLSL shaders in SwiftShader in Google Chrome prior to 72.0.3626.81 allowed a remote						

~~Stack~~ Buffer Overflow

```c
int main() {
  char buf[8];
  read(0, buf, 1337);
  return 0;
}
```

```c
int main() {
  char buf[8];
  scanf("%s", buf);
  return 0;
}
```

```c
int main() {
  char buf[8];
  gets(buf);
  return 0;
}
```

```c
int hackme(char *data) {
  char buf[8];
  memcpy(buf, data, 1337);
  return 0;
}
```

```c
int hackme(char *data) {
  char buf[8];
  strcpy(buf, data);
  return 0;
}
```

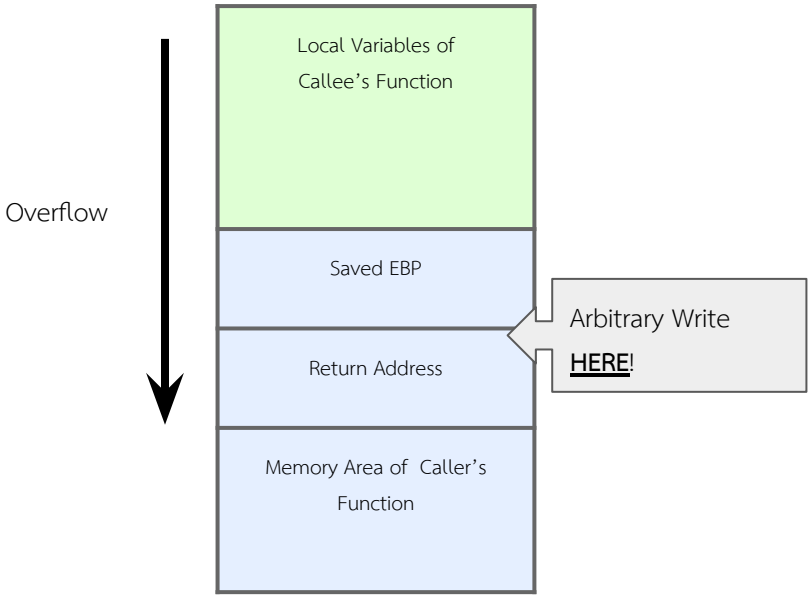
```c
int hackme(char *name) {
  char buf[8] = "Hello: ";
  strcat(buf, name);
  return 0;
}
```

~~Stack~~ Buffer Overflow

คำอธิบาย

Buffer Overflow: เกิดเมื่อมีการเขียนข้อมูลเกินขนาดพื้นที่หน่วยความจำที่จัดไว้ ทำให้ข้อมูลล้นและอาจทำให้โปรแกรมทำงานผิดพลาดหรือถูกโจมตีได้

Memory Arbitrary Write

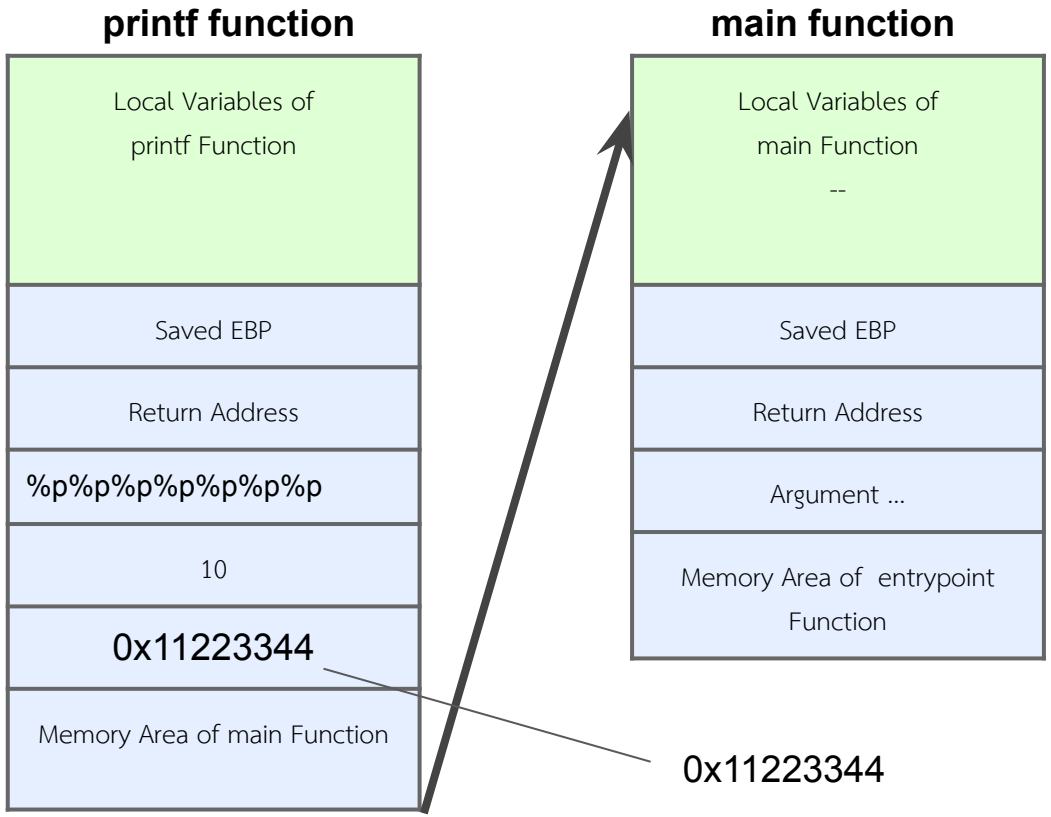


Memory Arbitrary Read

Can it happen ?

```
1 int main() {
2   char buf[8];
3   read(0, buf, 1337);
4   printf("%s", buf);
5   return 0;
6 }
```

Absolutely YES !



Format String

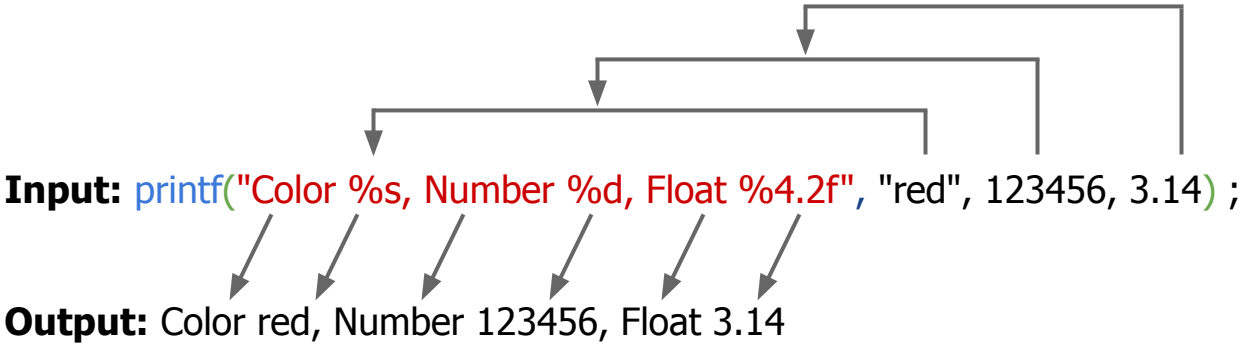
Direct Reference แสดงผล Argument ปัจจุบันโดยตรง

%d แสดงผลเป็นเลขฐาน 10
%x แสดงผลเป็นเลขฐาน 16
%p แสดงผลเป็นรูปแบบของ Address
%h แสดงผลเป็นเลขฐาน 10 (2 bytes)
…

Pointer Reference แสดงผลหรือเขียนค่าที่ Argument ปัจจุบันชี้ไปหา

%s แสดงผลเป็น null-terminated string
%n เขียนจำนวนที่

Format String

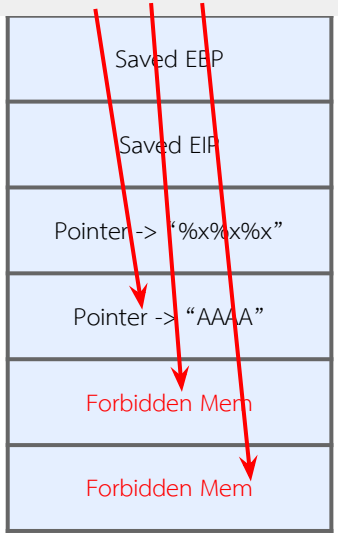


วิธีอ่าน
%s – แทนที่ด้วย **String** (เป็น "red")
%d – แทนที่ด้วย **Integer** (เป็น 123456)
%4.2f – แทนที่ด้วย **Float** โดยมี 2 ตำแหน่งทศนิยม (เป็น 3.14)

Format String

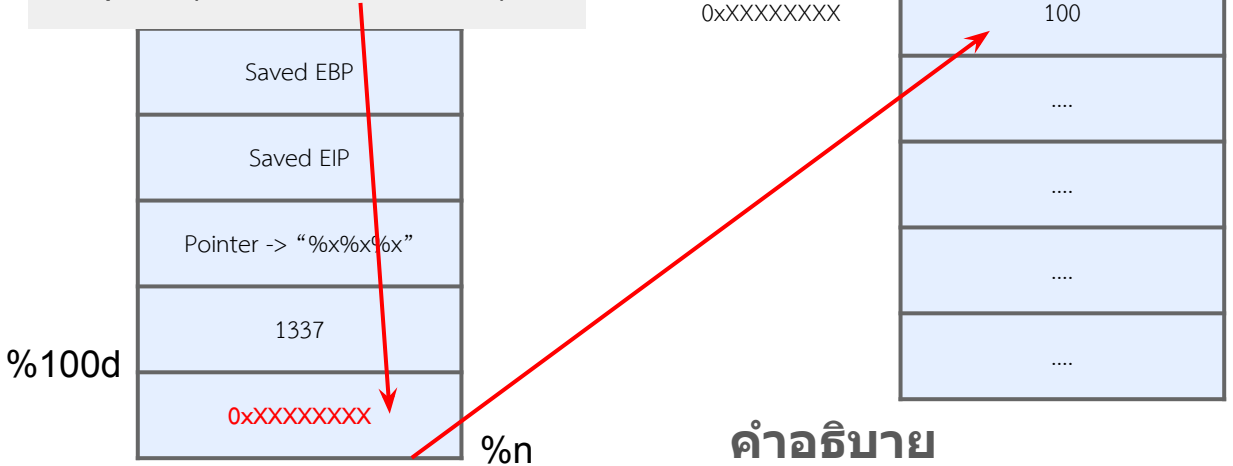
Memory Arbitrary Read

```
printf("%x%x%x", "AAAA")
```



Memory Arbitrary Write

```
printf("%100d%n", 1337)
```

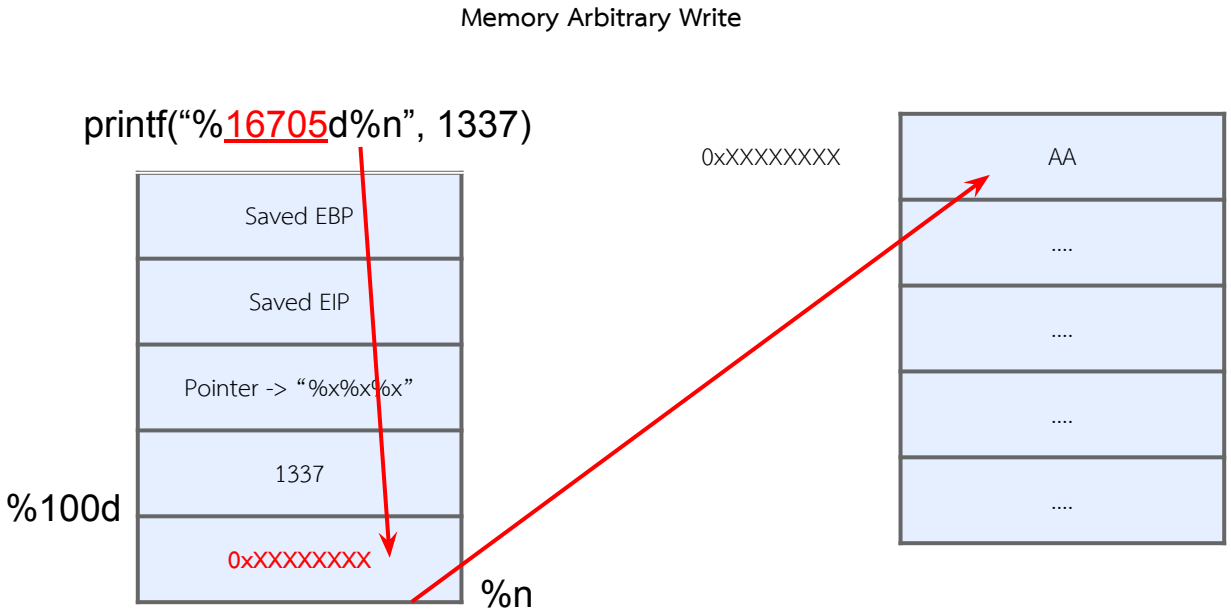


<space 96 ตัว>1337

คำอธิบาย

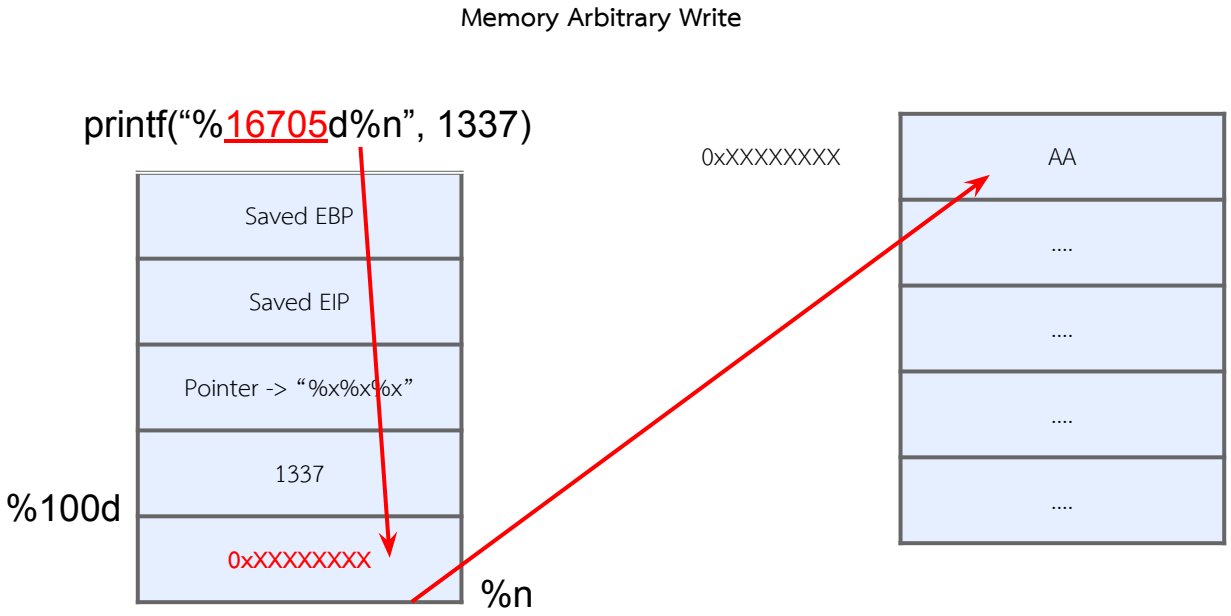
Format String: ช่องโหว่ที่เกิดเมื่อมีการใช้ฟังก์ชันแสดงผล เช่น **printf** โดยไม่ได้ควบคุมรูปแบบการส่งข้อมูล ทำให้ผู้โจมตีสามารถควบคุมรูปแบบและแทรกข้อมูลที่ไม่พึงประสงค์ลงในหน่วยความจำได้

Format String - Pop Quiz



Format String - Pop Quiz

16,709 (decimal)
คือ
0x4141 (hex)
คือ
AA (ASCII)



Format String

```c
int main(){
    char input[20];
    scanf("%s ", input);
    printf(input);
    return 0;
}
```

Memory Arbitrary Read

```
kali@kali:~/work/pwn_by_bongtrop$ ./format
AAAA.%x.%x.%x.%x.%x.%x.%x
AAAA.a.0.0.ffffff80.40.41414141.252e7825
```

```
kali@kali:~/work/pwn_by_bongtrop$ ./format
AAAA.%6$x
AAAA.41414141
```

```c
int main()
{
  int val;
  printf("blah %n blah\n", &val);
  printf("val = %d\n", val);
  return 0;
}
```

Memory Arbitrary Write

```
kali@kali:~/work/pwn_by_bongtrop$ ./format-write
blah  blah
val = 5
```

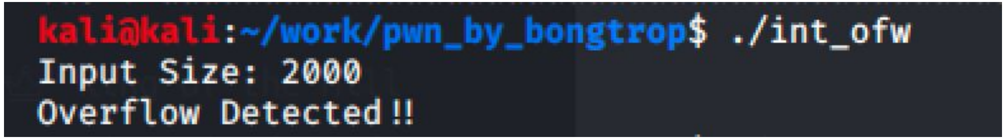
และสามารถเขียนค่าลงตำแหน่งใด ๆ ได้ด้วย %Xc%Y$n ด้วยขนาดความยาวของข้อมูลก่อนหน้า เช่น %30x%10$n คือการเขียนค่าที่ตำแหน่งที่ 10 ด้วยค่า 30 (%30x มีขนาด 30 ตัว)

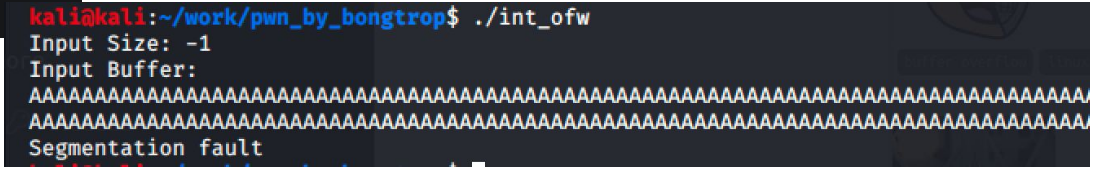
Integer Overflow

```
int main(){
        int size;
        char buf[BUF_SIZE];
        printf("Input Size: ");
        scanf("%d", &size);
        if(size >= BUF_SIZE){
                printf("Overflow Detected!!\n");
                exit(1337);
        }
        printf("Input Buffer: \n");
        read(0, buf, size);
        return 0;
}
```

คำอธิบาย

Integer Overflow: เกิดเมื่อค่าตัวเลขเกินขอบเขตที่สามารถเก็บได้ในประเภทข้อมูลที่กำหนด ทำให้ค่าตัวเลข "ล้น" และวนกลับไปเริ่มต้นใหม่ ซึ่งอาจทำให้เกิดผลลัพธ์ที่ไม่คาดคิดหรือช่องโหว่ด้านความปลอดภัยได้





Array Index out of Bounds

คำอธิบาย

Array Index Out of Bounds: เกิดเมื่อมีการเข้าถึงตำแหน่งใน Array ที่อยู่นอกขอบเขตที่กำหนดไว้ ซึ่งอาจทำให้เกิดข้อผิดพลาดในโปรแกรมหรือถูกโจมตีเพื่อควบคุมหน่วยความจำได้

Memory Arbitrary Read

```c
int config[SIZE] = {0};

int get_config(int pos) {
  if (pos >= SIZE) exit(1337);
  return config[pos];
}
```

Memory Arbitrary Write

```c
int config[SIZE] = {0};

int set_config(int pos, int val) {
  if (pos >= SIZE) exit(1337);
  config[pos] = val;
  return 0;
}
```

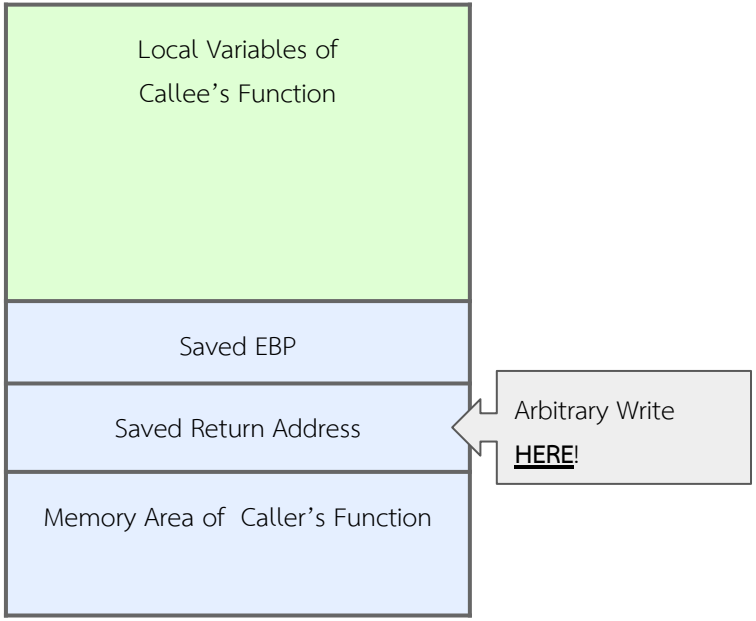
Program Flow Hijacking

แก้ EIP/RIP register เป็นค่าที่เราต้องการ ตัวอย่างเช่น

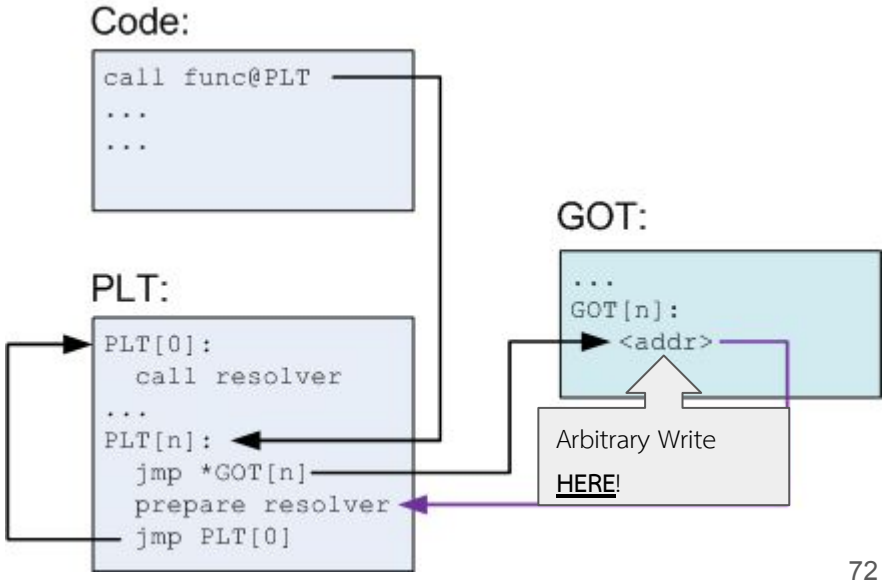
คำอธิบาย

Program Flow Hijacking: เป็นการโจมตีที่ผู้โจมตีสามารถควบคุมลำดับการทำงานของโปรแกรม โดยการแก้ไขการกระโดดหรือการเรียกใช้ฟังก์ชันในโค้ด เพื่อให้โปรแกรมรันโค้ดที่ไม่พึงประสงค์ เช่น การใช้ประโยชน์จาก Buffer Overflow หรือ Return-Oriented Programming (ROP)

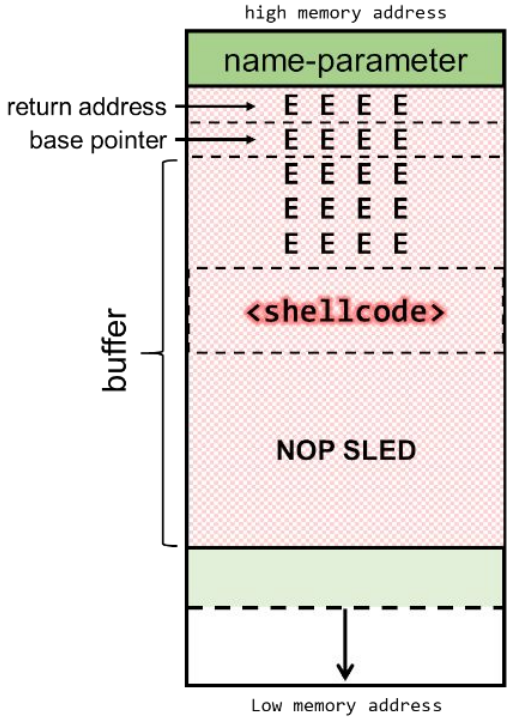
Arbitrary write to function's return address



Arbitrary write to function pointer



Shellcode



Shellcode คือ โค้ดขนาดเล็กที่ถูกออกแบบมาเพื่อใช้ในการโจมตีระบบ โดยมักถูกฝังลงในช่องโหว่ เช่น Buffer Overflow เพื่อให้ผู้โจมตีสามารถควบคุมการทำงานของระบบหรือเรียกใช้งานคำสั่งอันตราย เช่น เปิด Shell หรือรันคำสั่งอื่น ๆ บนเครื่องเป้าหมาย Shellcode มักถูกเขียนในภาษา Assembly เพื่อให้มีขนาดเล็กและทำงานได้อย่างรวดเร็ว.

```
coen@kali:/tmp/coen$ ./envexec.sh buf $(python -c 'print "\x90" * 63 + "\x31\xc0\x50\x68\x2f\
x89\xe1\xb0\x0b\xcd\x80" + "\x6c\xfd\xff\xbf" * 5')
Welcome ������������������������������������������������������1̀Ph//shh/biǹ̀P̀̀S̀̀̀

# whoami
root
# 
```

ที่มา:
https://www.coengoedegebure.com/buffer-overflow-attacks-explained/

ลองทำแลป !

Agenda (Day 1)

เวลา	รายละเอียด
09.15 - 09.45	ความรู้เบื้องต้นเกี่ยวกับ CTF
09.45 - 10.30	Network Security
10.30 - 10.45	พักเบรก
10.45 - 12.00	Web Application Security
12.00 - 13.00	พักรับประทาน อาหารกลางวัน
13.00 - 14.30	Digital Forensics
14.30 - 14.45	พักเบรก
14.45 - 16.00	Pwnable & Reverse Engineering
16.00 - 18.00	เข้าห้องพัก
18.00 - 19.00	รับประทานอาหารเย็น
19.00 - 21.00	เสวนาแนะแนวเส้นทางอาชีพ

Thank you !!