

How to be a malware analyst/research

Azlan Mukhtar

Co-Founder Eraxen PLT

About me

- Ex-Malware Analyst/Researcher
- Coder and Reverser
- Wannabe entrepreneur
- Programming languages enthusiast

Who is malware analyst

- Somebody who analyzes and studies malware by using some techniques and tools
- A cool guy



What kind of analyst

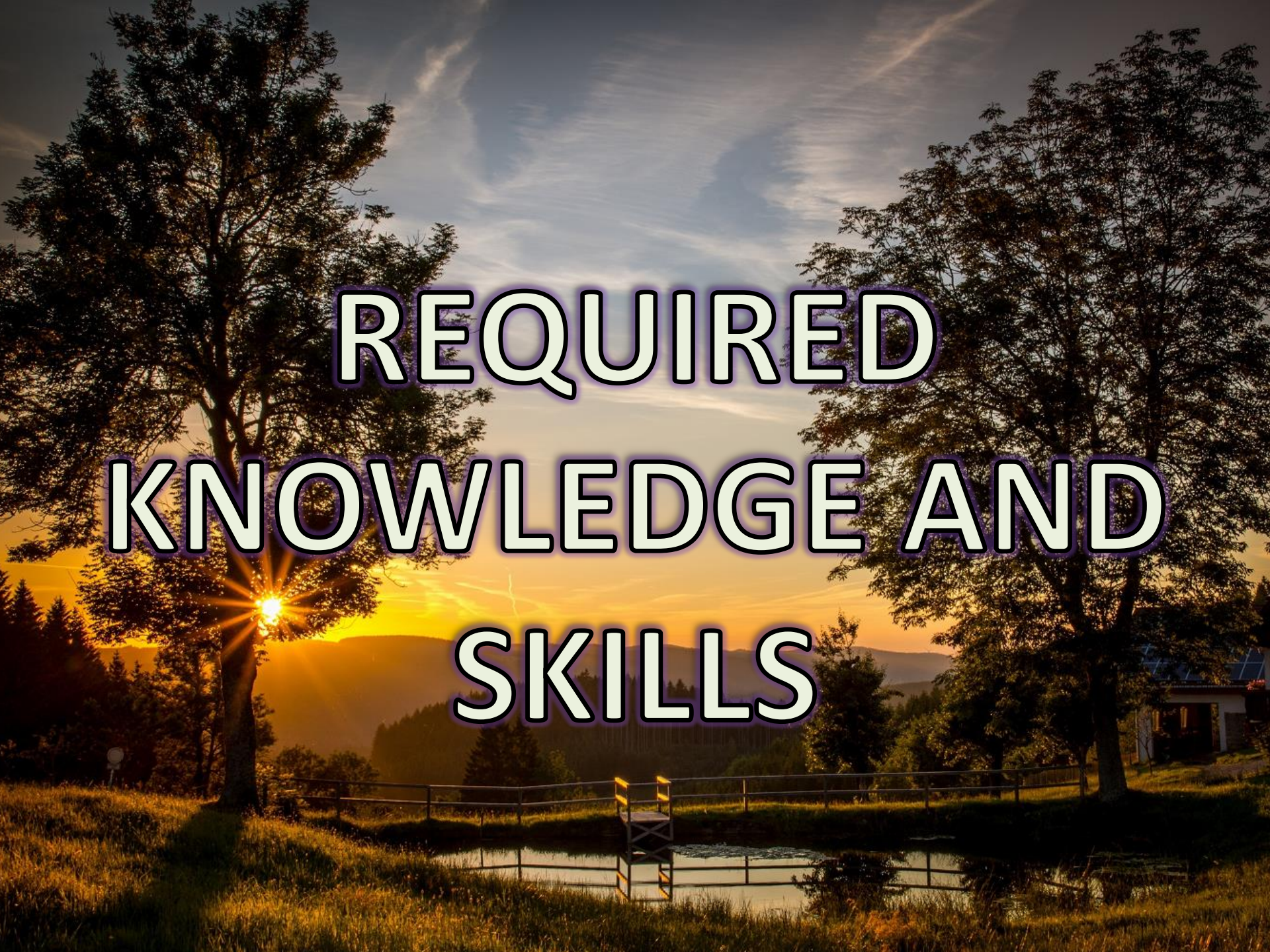
- Two categories of malware analyst
- First:
 - Only do dynamic analysis
 - Using dynamic analysis tools such Sysinternals Suite, sandboxes, etc
- Second:
 - All of above
 - Analyze deeper, static analysis and debugging

How to be malware analyst

- Required knowledge and skills
- Learning paths



MALWARE

A scenic landscape at sunset. The sun is low on the horizon, creating a warm orange glow. Two large trees frame the scene on the left and right. In the foreground, there is a grassy field with a wooden fence and a small pond. The text "REQUIRED KNOWLEDGE AND SKILLS" is overlaid in the center in a large, white, outlined font.

REQUIRED KNOWLEDGE AND SKILLS

Quiz 1

```
void function1(void *a, void *b, size_t n)
{
    char *x = (char *)b;
    char *y = (char *)a;

    for (int i=0; i<n; i++) {
        y[i] = x[i];
    }
}
```


Quiz 2

```
01061020  PUSH  EBP
01061021  MOV   EBP,ESP
01061023  PUSH  ECX
01061024  MOV   DWORD PTR SS:[EBP-4],0
0106102B  JMP   SHORT 01061036
0106102D  MOV   EAX,DWORD PTR SS:[EBP-4]
01061030  ADD   EAX,1
01061033  MOV   DWORD PTR SS:[EBP-4],EAX
01061036  MOV   ECX,DWORD PTR SS:[EBP-4]
01061039  CMP   ECX,DWORD PTR SS:[EBP+10]
0106103C  JAE   SHORT 01061050
0106103E  MOV   EDX,DWORD PTR SS:[EBP+8]
01061041  ADD   EDX,DWORD PTR SS:[EBP-4]
01061044  MOV   EAX,DWORD PTR SS:[EBP+0C]
01061047  ADD   EAX,DWORD PTR SS:[EBP-4]
0106104A  MOV   CL,BYTE PTR DS:[EAX]
0106104C  MOV   BYTE PTR DS:[EDX],CL
0106104E  JMP   SHORT 0106102D
01061050  MOV   ESP,EBP
01061052  POP   EBP
01061053  RETN
```


Answer

```
void myMemCpy(void *dest, void *src, size_t n)
{
    // Typecast src and dest addresses to (char *)
    char *csrc = (char *)src;
    char *cdest = (char *)dest;

    // Copy contents of src[] to dest[]
    for (int i=0; i<n; i++) {
        cdest[i] = csrc[i];
    }
}
```

```
1 void myMemCpy(void *dest, void *src, int n)
2 {
3     // Typecast src and dest addresses to (char *)
4     char *csrc = (char *)src;
5     char *cdest = (char *)dest;
6
7     // Copy contents of src[] to dest[]
8     for (int i = 0; i < n; i++) {
9         cdest[i] = csrc[i];
10    }
11 }
12
13
14
15
16 // https://godbolt.org/
```

#1 with x86-64 clang 3.9.0 x
x86-64 clang 3.9.0 compiler options...

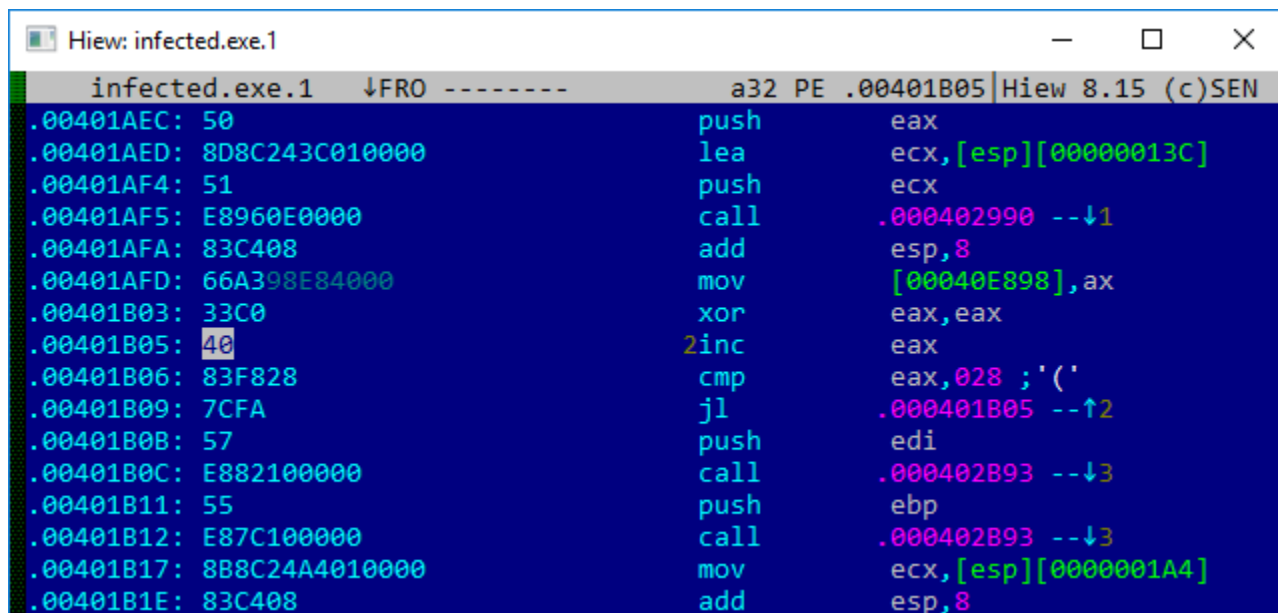
11010	.LX0:	.text	//	Intel	A	A	A	+
1	myMemCpy(void*, void*, int):							#
	@myMemCpy(void*, void*, int)							
2	push	rbp						
3	mov	rbp, rsp						
4	mov	qword ptr [rbp - 8], rdi						
5	mov	qword ptr [rbp - 16], rsi						
6	mov	dword ptr [rbp - 20], edx						
7	mov	rsi, qword ptr [rbp - 16]						
8	mov	qword ptr [rbp - 32], rsi						
9	mov	rsi, qword ptr [rbp - 8]						
10	mov	qword ptr [rbp - 40], rsi						
11	mov	dword ptr [rbp - 44], 0						
12	.LBB0_1:							# =>This Inner Loop
	Header: Depth=1							
13	mov	eax, dword ptr [rbp - 44]						
14	cmp	eax, dword ptr [rbp - 20]						
15	jge	.LBB0_4						
16	movsxd	rax, dword ptr [rbp - 44]						
17	mov	rcx, qword ptr [rbp - 32]						
18	mov	dl, byte ptr [rcx + rax]						
19	movsxd	rax, dword ptr [rbp - 44]						
20	mov	rcx, qword ptr [rbp - 40]						
21	mov	byte ptr [rcx + rax], dl						
22	mov	eax, dword ptr [rbp - 44]						
23	add	eax, 1						
24	mov	dword ptr [rbp - 44], eax						
25	jmp	.LBB0_1						
26	.LBB0_4:							
27	pop	rbp						

Required knowledge

- Basic computer architecture
- Operating System
- Native OS API
- C/C++ programming
- Assembly language programming
- Executable file format
- Scripting (Python, Javascript, VB Script)

Required knowledge – cont.

- Reverse engineering
 - **[Program] reading comprehension**
 - Disassembling, decompiling, and debugging techniques



```
Hiew: infected.exe.1
infected.exe.1  ↓FRO ----- a32 PE .00401B05 | Hiew 8.15 (c)SEN
.00401AEC: 50          push    eax
.00401AED: 8D8C243C010000    lea     ecx,[esp][00000013C]
.00401AF4: 51          push    ecx
.00401AF5: E8960E0000    call    .000402990 --↓1
.00401AFA: 83C408       add     esp,8
.00401AFD: 66A398E84000  mov     [00040E898],ax
.00401B03: 33C0        xor     eax,eax
.00401B05: 40          inc     eax
.00401B06: 83F828       cmp     eax,028 ;'('
.00401B09: 7CFA        j1      .000401B05 --↑2
.00401B0B: 57          push    edi
.00401B0C: E882100000    call    .000402B93 --↓3
.00401B11: 55          push    ebp
.00401B12: E87C100000    call    .000402B93 --↓3
.00401B17: 8B8C24A4010000 mov     ecx,[esp][0000001A4]
.00401B1E: 83C408       add     esp,8
```


Pressing F8 in OllyDBG



A scenic landscape featuring a calm pond in the center, reflecting the sky. To the left, a small dark barn is partially visible behind a large tree. A dirt road with tire tracks leads from the bottom left towards the pond. The foreground is filled with green grass and some low-lying shrubs. A wire fence runs across the middle ground. The sky is a vibrant blue with scattered white clouds. The text "Learning Paths" is overlaid in the center in a white, outlined font.

Learning Paths

How to start

- Start with C programming
 - C programming language
 - Learn how to use debugger to debug your C program
- Assembly language for the platform architecture
 - x86 or ARM
- Scripting (Python, Javascript)

Operating System Internal

- Process
- Virtual memory
- Executable file format

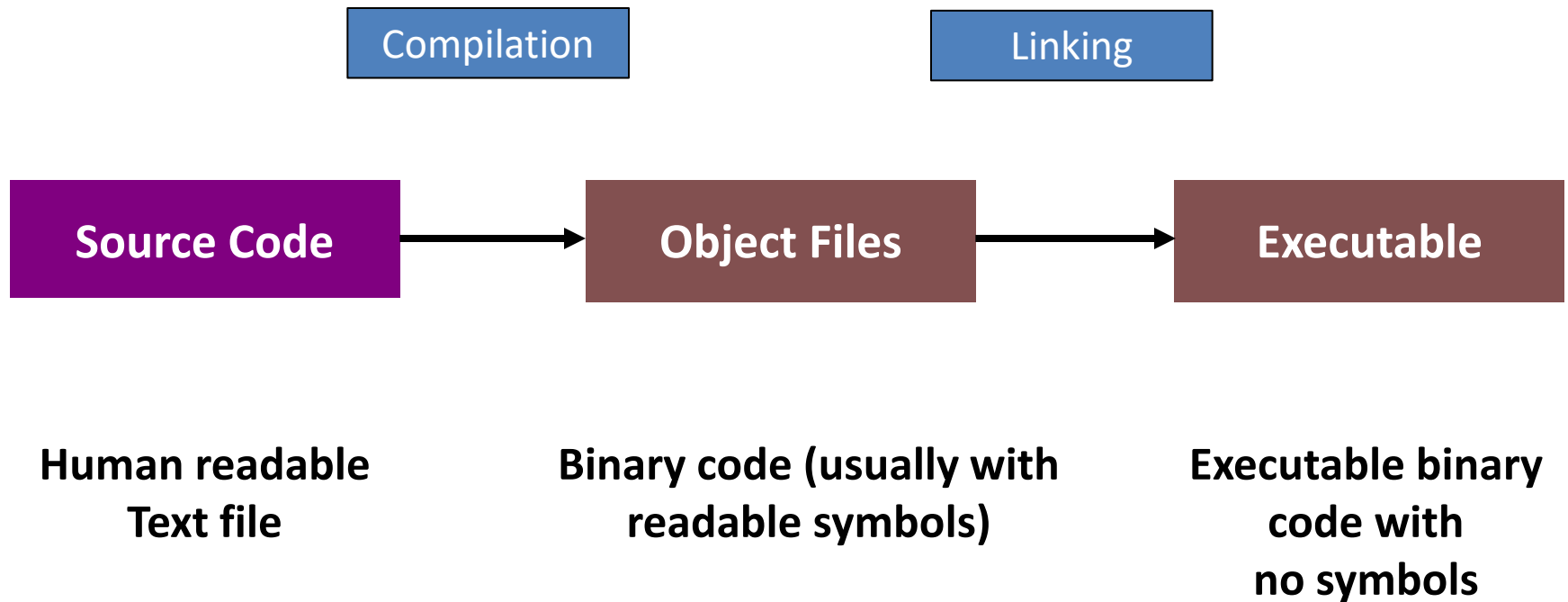
Reverse engineering

- Reverse Engineering is also known as RE or RCE
 - RE: Reverse Engineering
 - RCE: Reverse Code Engineering
- RE is the process of understanding an existing product. In our case, malicious software.
- Malware analysis and security research often involves RE

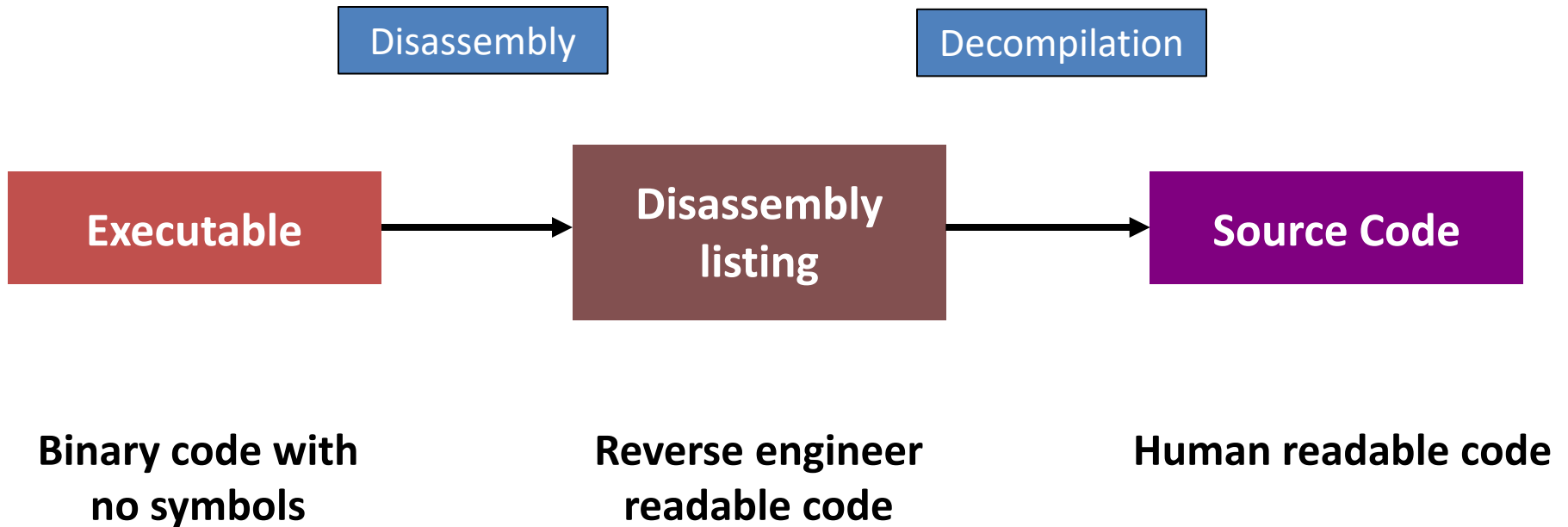
Use cases for RE

- Malware analysis
- Security / Vulnerability Research
- Driver development
- Compatibility fixes
- Legacy application support

Compilation process



Decompilation process



Tools of the Trade

- Hex Editor / Viewer
 - HIEW / BIEW
- Disassembler
 - IDA Pro
- Debugger
 - WinDBG
 - OllyDbg
- **Your own tools**

Learn the right way of reversing

- **Learn how to read code for program comprehension**
- Use static analysis tools
 - IDA Pro
- Smart debugging
 - No need waste time analyze wrong codes
- Get a good mentor

The challenges of Malware Reversing

- Obfuscation
- Packed and obfuscated malware
- The anti
 - Anti-debug
 - Anti-dump
 - Anti-disassembly
- Other clever techniques to prevent reversing

OS Native API

```
FILE *fp;  
fp = fopen(const char *filename, const char *mode);
```

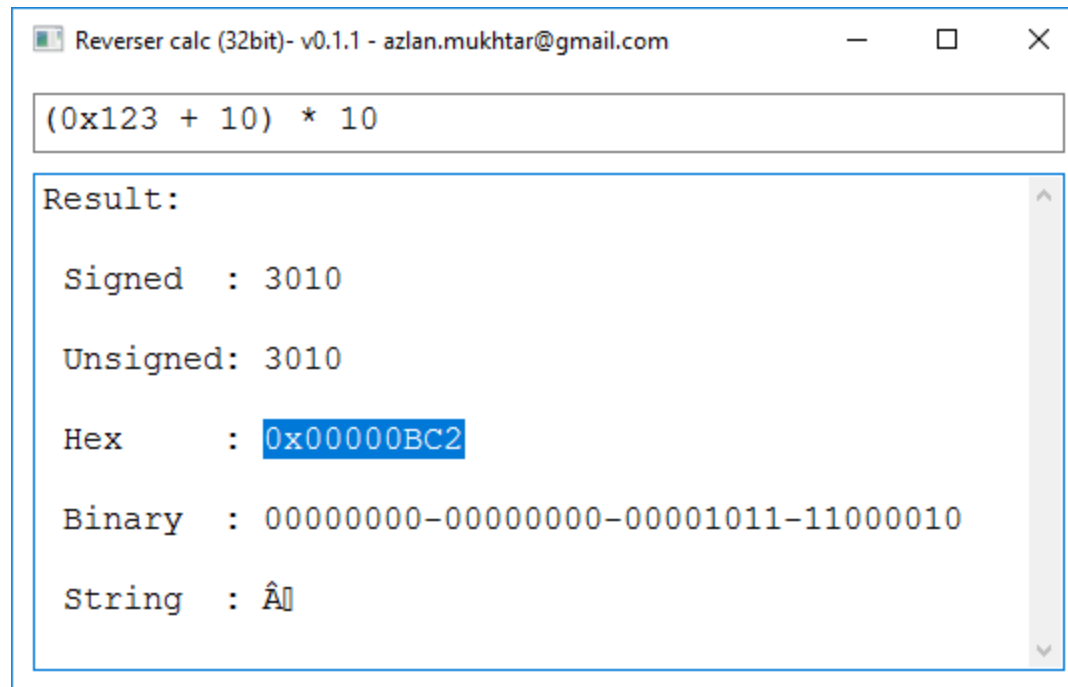
VS

```
HANDLE hFile;  
hFile = CreateFile("filename", GENERIC_READ/WRITE, ..., NULL);
```

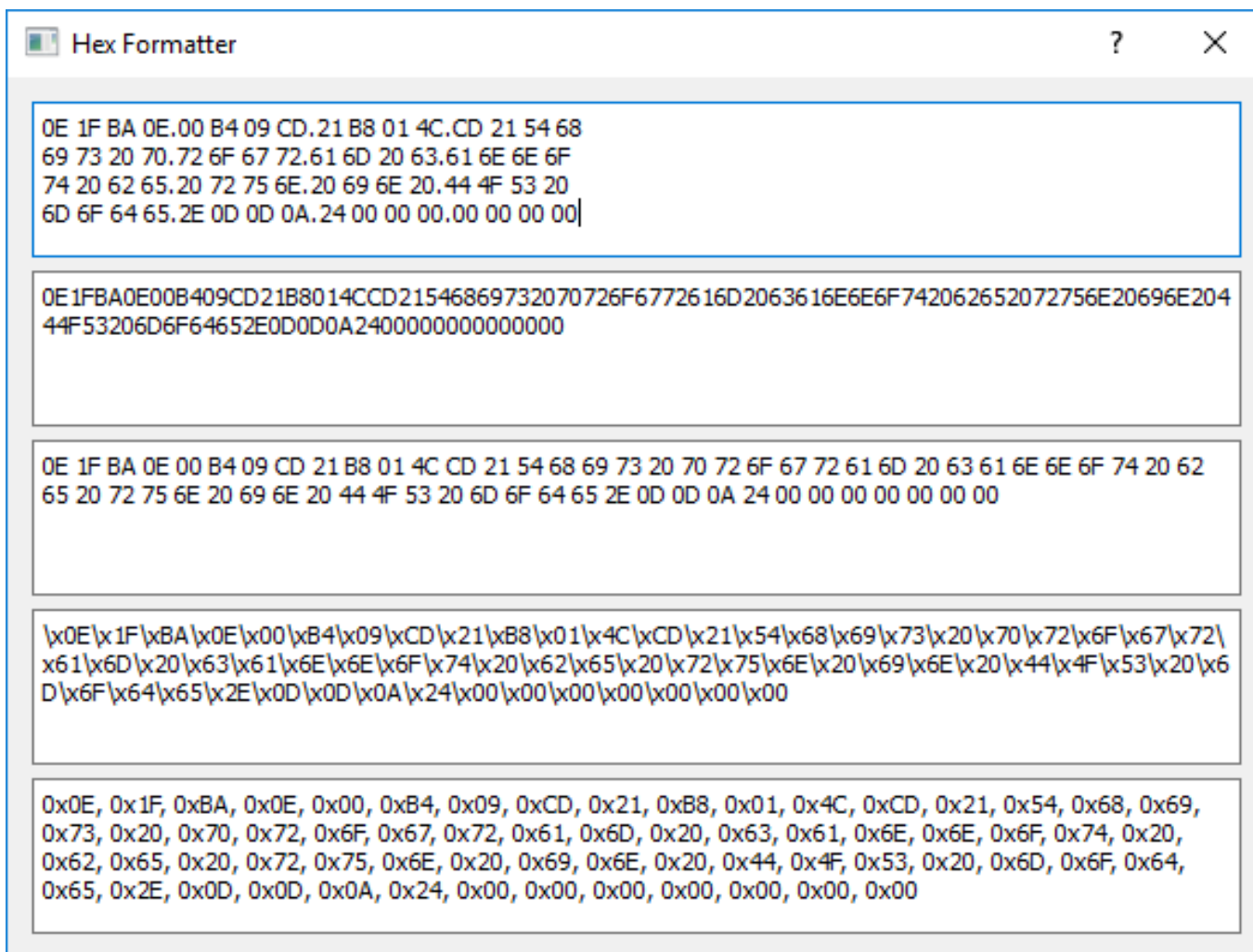
VS

```
int fd = open(const char *pathname, int flags, mode_t mode);
```

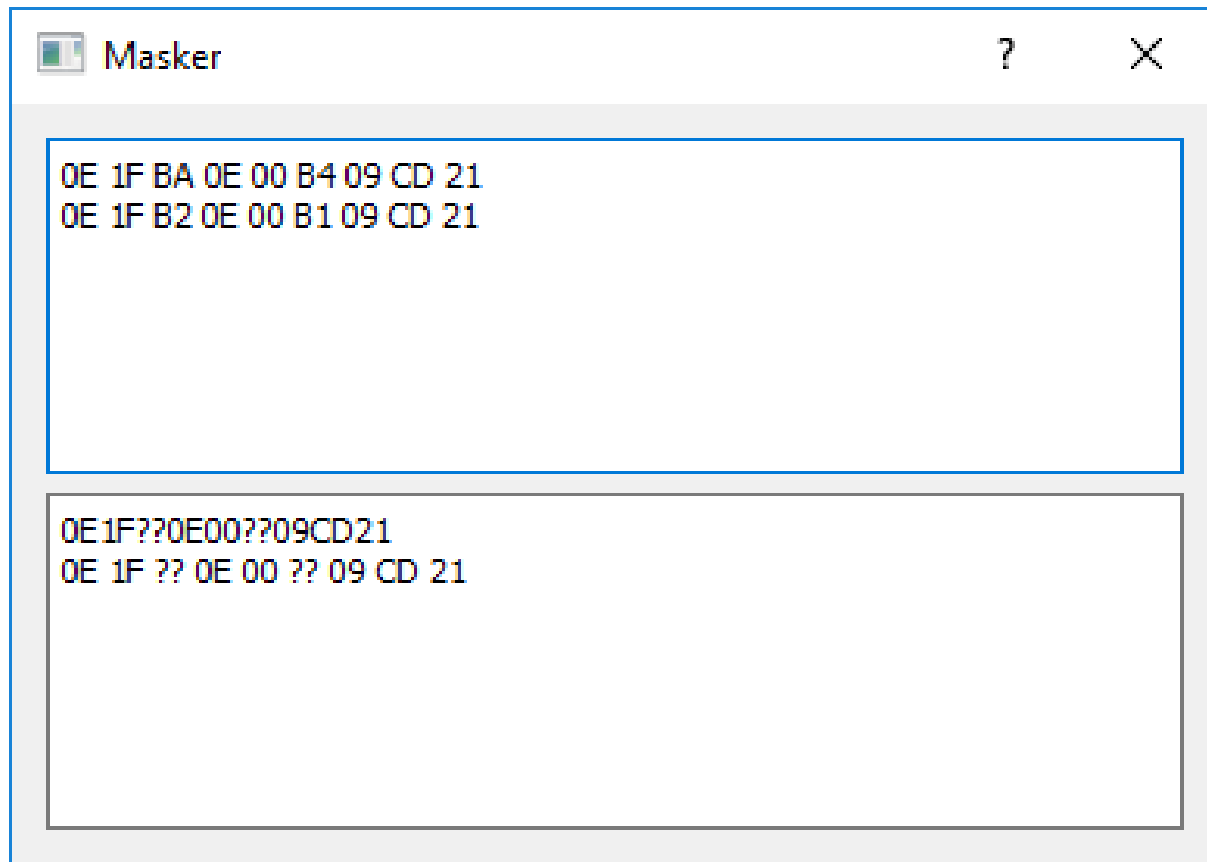
Code your own tools



Code your own tools – cont.



Code your own tools – cont.



What else

- Develop algorithmic thinking
 - Project Euler - <https://projecteuler.net/>
- Always be coding and reversing
- Mastery take years, don't take shortcuts
- Do a lot of practice
- Participate reverse engineering challenges (CTF)

References

- Books
 - The C Programming Language, 2nd Edition
 - Windows via C/C++, 5th Edition
 - Guide to Assembly Language: A Concise Introduction
 - Windows Internals, 6th Edition
 - IDA Pro Book, 2nd Edition
 - Practical Malware Analysis

Q & A



Thank you

- Contact: azlan@eraxen.com