JPCERT CC®

Understanding Malware

2015/08/14 Security Camp 2015 13-D, 14-D

JPCERT/CC Analysis Center
You NAKATSURU

Notice

- These training materials are used for "Security Camp 2015" in Japan
  - Security training program for students to discover & nurture young talent
  - https://www.ipa.go.jp/jinzai/camp/ (Japanese only)
- The training course consists of the following 2 parts
  - Malware, Malware analysis basics, Static analysis basics
    - Learning basic knowledge for malware analysis
  - Malware analysis
    - Understanding details of malware samples using static analysis method
- The training mainly focuses on 32bit Windows malware
- Some slides have display problems due to animation
- Any questions and comments are welcome
  - Please contact us at aa-info@jpcert.or.jp

Agenda

- Basic Knowledge
- Malware Analysis
  - Simple HTTP Bot
  - Banking Trojan
- Bonus
  - Shellcode
  - MWS Cup
- Discussion

Objectives of This Session

Understanding malware

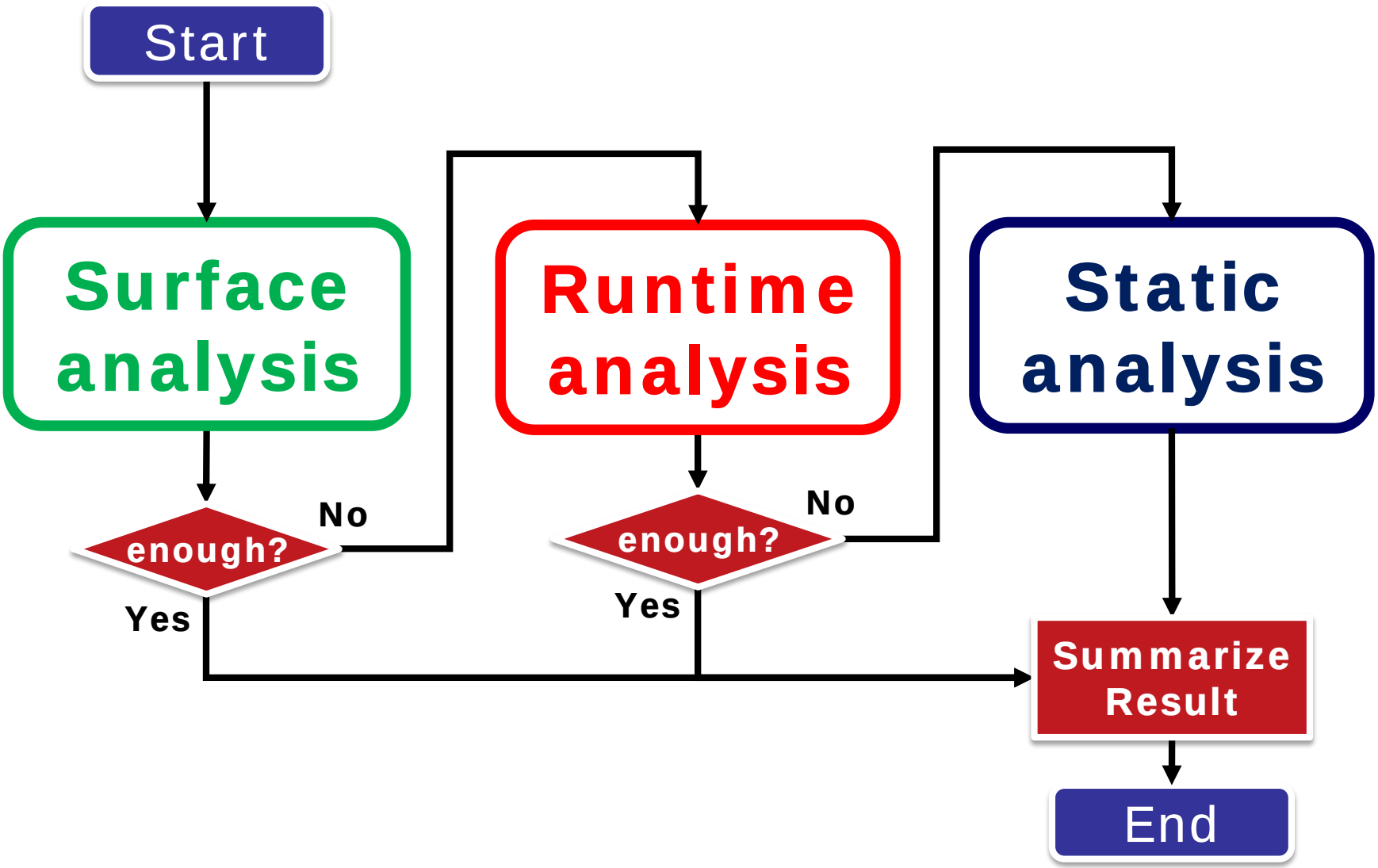
- Windows features used by malware
- Implementation of "real" malware
  - HTTP Bot
  - Banking Trojan

Understanding static analysis

- Difficulties and Challenges

Windows Malware Analysis

(recap) Malware Analysis Flow



(recap) Analysis Process Comparison

	Surface analysis	Runtime analysis	Static analysis
Overview	Retrieve surface information from targets **without execution**	**Execute samples** and monitor its behavior	**Read codes** in binary files and understand its functionality
Output	- Hash values - Strings - File attributes - Packer info - Anti-virus detection info	Activity of - File system - Registry - Process - Network	Malware's functionality e.g. - Bot commands - Encode/decode methods
Security risk	Low	**High**	Moderate
Analysis coverage	Low	Moderate	**High**

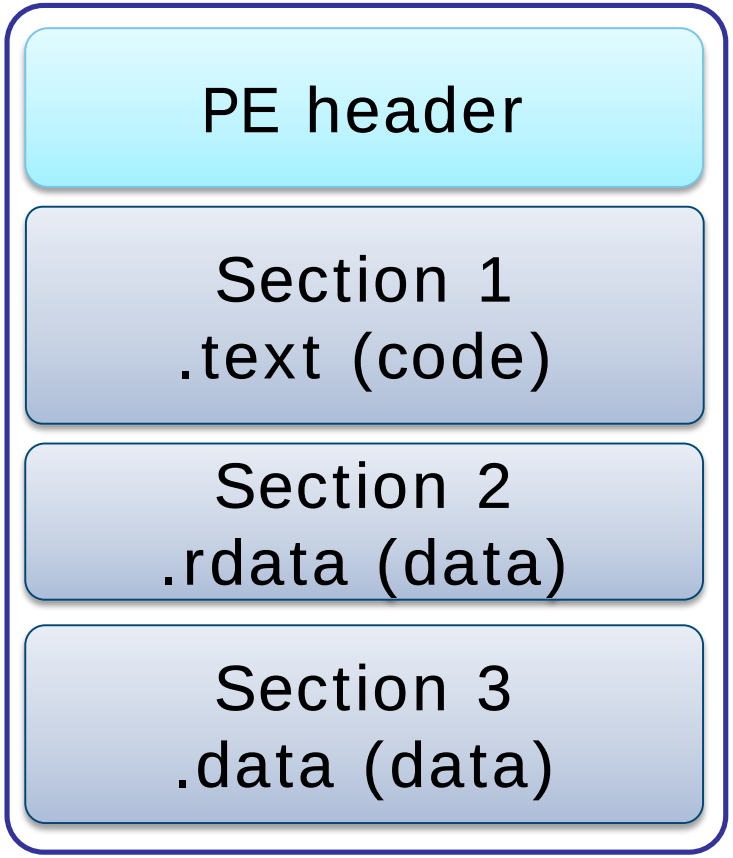
(recap) Static Analysis Tools

Category	Name	Description
Disassembler	IDA	Disassembles more than 50 architectures
Decompiler	Hex-rays Decompiler	x86/ARM binary to C source code
	VB Decompiler	Visual Basic binary to Visual Basic source code
	.NET Reflector	.NET binary to .NET source code
Debugger	OllyDbg	World famous X86 debugger
	Immunity Debugger	Python familiar x86 debugger

BASIC KNOWLEDGE

PE (Portable Executable) File Format

- https://msdn.microsoft.com/en-us/windows/hardware/gg463119.aspx
- Consists of headers and multiple sections, will be extended on memory
  - —Header: File Information
    - Entry point
    - Timestamp
    - Section's info
    - etc.
  - —Section: Byte code, data



EXE & DLL

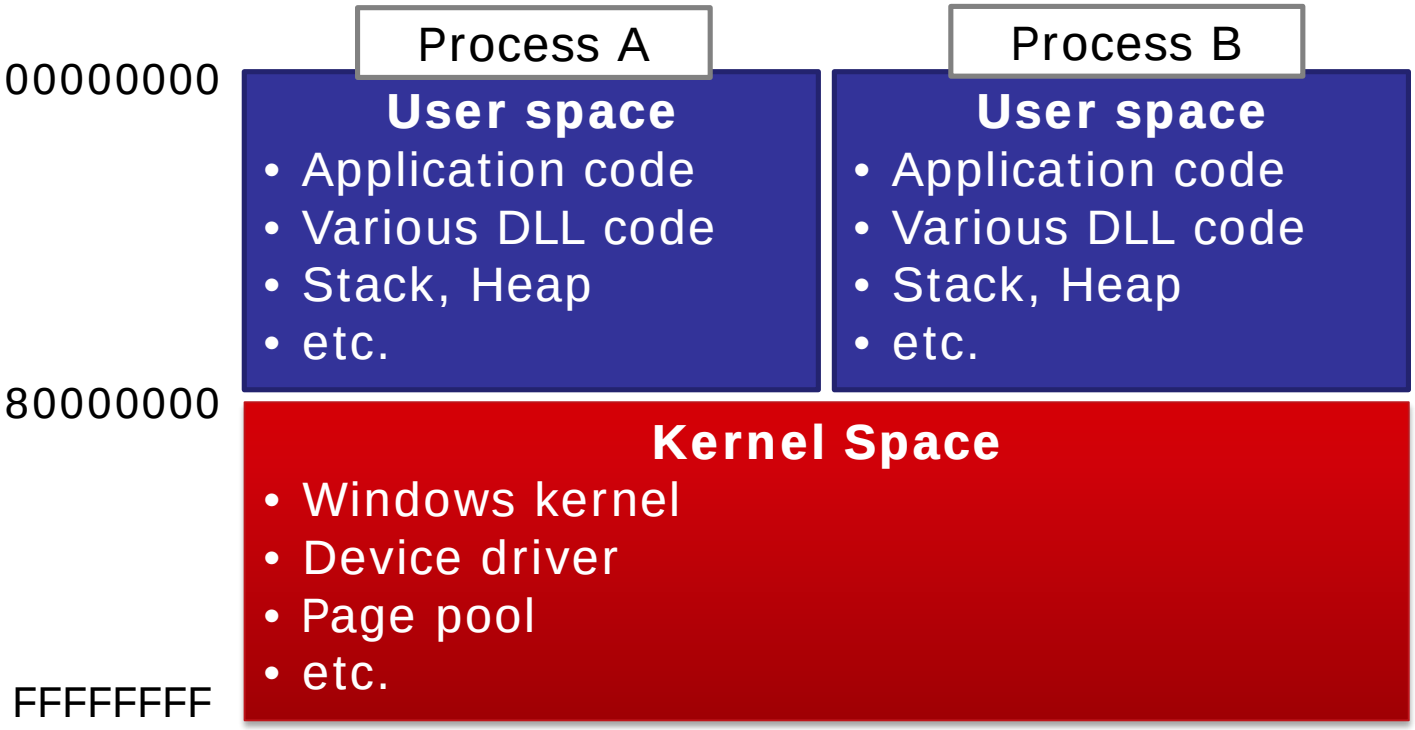
- "EXE" and "DLL" are 2 most common file types in PE (Portable Executable) file format
  - —"Characteristics" of PE header

	EXE 	DLL 
File Format	Portable Executable	
Summary	Independent application file	Collection of functions as shared library
Example	explorer.exe, iexplore.exe	kernel32.dll, shell32.dll

Execute timing	• Main function
	  • when the file is executed
		• Main function
		  • when the DLL is loaded/unloaded
		  • when a thread start/exit
		• Exported function
		  • when is called

Process & Virtual Memory

- 4GB per process (32bit Windows)
  - —User space 2GB
    - allocated for each process, able to access each other
  - —Kernel space 2GB
    - shared with all processes



Finding Main Function

- Windows executable binary file will be started with initial processing to launch the process
- To find main function

Understand its initialization routine

- Compile & Disassemble your program

Use tools

- OllyDbg / Immunity Debugger
- IDA Starter/Pro

Use your sixth sense

- Based on your experience

Important Points

Do not read everything

- Time is money
- Remember "efficient code analysis"

Analysis is not our purpose

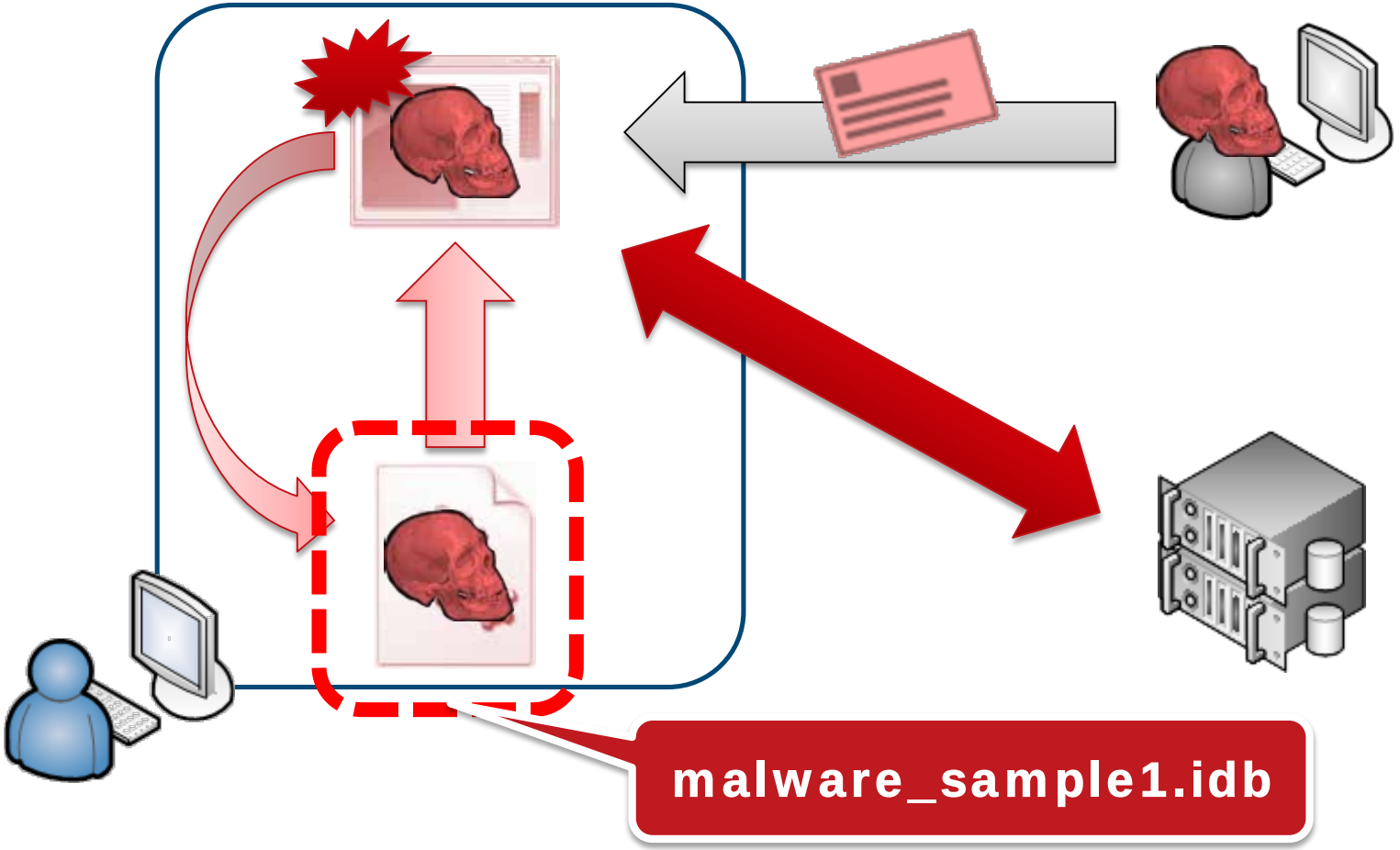
- Wrap up analysis results for incident response/information sharing
  - Analysis report/note
  - Commented IDB file

Malware Analysis

LET'S ANALYZE SIMPLE HTTP BOT

Analysis Target

- A kind of HTTP bot spread through mass emails



Exercise 1. Malware Analysis

i. Describe the following points of the target
   - —Details of each bot command
   - —Decode method
     - Try to decode malware_sample1_data.bin

ii. Make your IDB
   - —Fill in information that you analyzed

Point 1. AutoRun Function

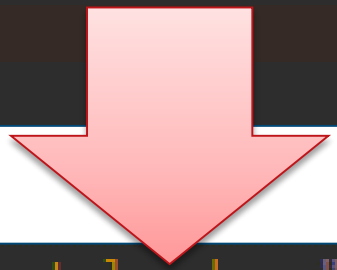
- Want to launch after rebooting the OS
  - —Copy itself into start up folder
  - —Add a registry entry to AutoRun part

Registry entries related to AutoRun
HKCU¥SOFTWARE¥Microsoft¥Active Setup¥Installed Components
HKCU¥SOFTWARE¥Microsoft¥Windows NT¥CurrentVersion¥Windows¥Run
HKCU¥SOFTWARE¥Microsoft¥Windows NT¥CurrentVersion¥Winlogon¥Shell
HKCU¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Run
HKCU¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥RunOnce
etc.

Point 2. Hiding API name

- Getting API address using GetProcAddress

```
push    offset aInternetcloseh ; "InternetCloseHandle"
push    offset ModuleName ; "wininet.dll"
call    ds:GetModuleHandleA
push    eax             ; hModule
call    ds:GetProcAddress
push    esi
call    eax
test    ebx, ebx
```



```
push    offset aInternetcloseh ; "InternetCloseHandle"
push    offset ModuleName ; "wininet.dll"
call    ds:GetModuleHandleA
push    eax             ; hModule
call    ds:GetProcAddress
push    esi             ; hInternet
call    eax             ; InternetCloseHandle
test    ebx, ebx
```

Point 3. HTTP Communication

- There are many ways to communicate using HTTP

WinINet APIs

- InternetOpen, HttpSendRequest, ...

WinSock APIs

- socket, connect, send, recv, ...

WinHTTP APIs

- WinHttpConnect, WinHttpSendRequest, ...

etc.

- URLDownloadToFile, ...

Point 4. Encoding (Obfuscation)

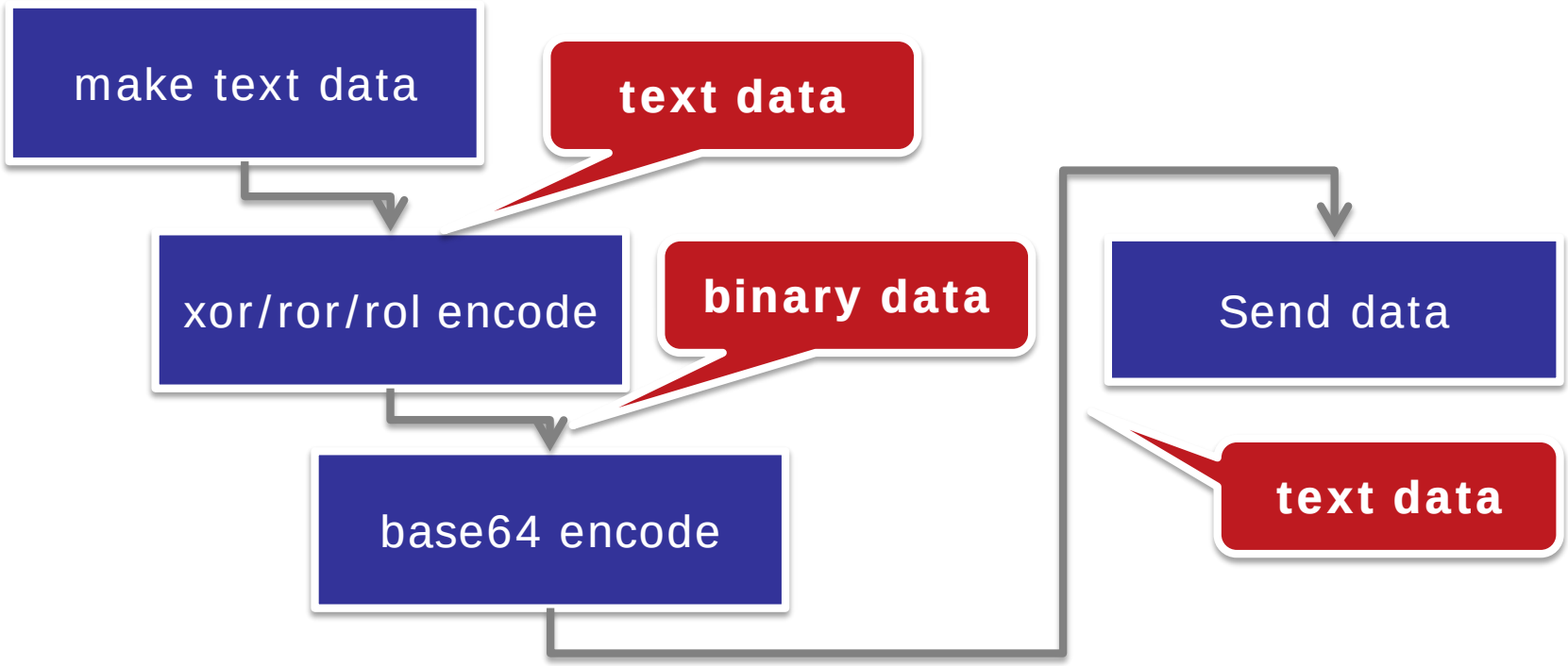
- Encode (encrypt) data to avoid being easily found
  - —Strings stored in the binary
    - File name, Registry entry name, Server address
  - —Packet

- Various methods are available

Method	Example
xor (exclusive or)	'a' ^ 0x05 = 'd'
ror/rol (rotate right/left)	rol 'a', 1 = 0xC2
base64	-
RC4	-
AES	-

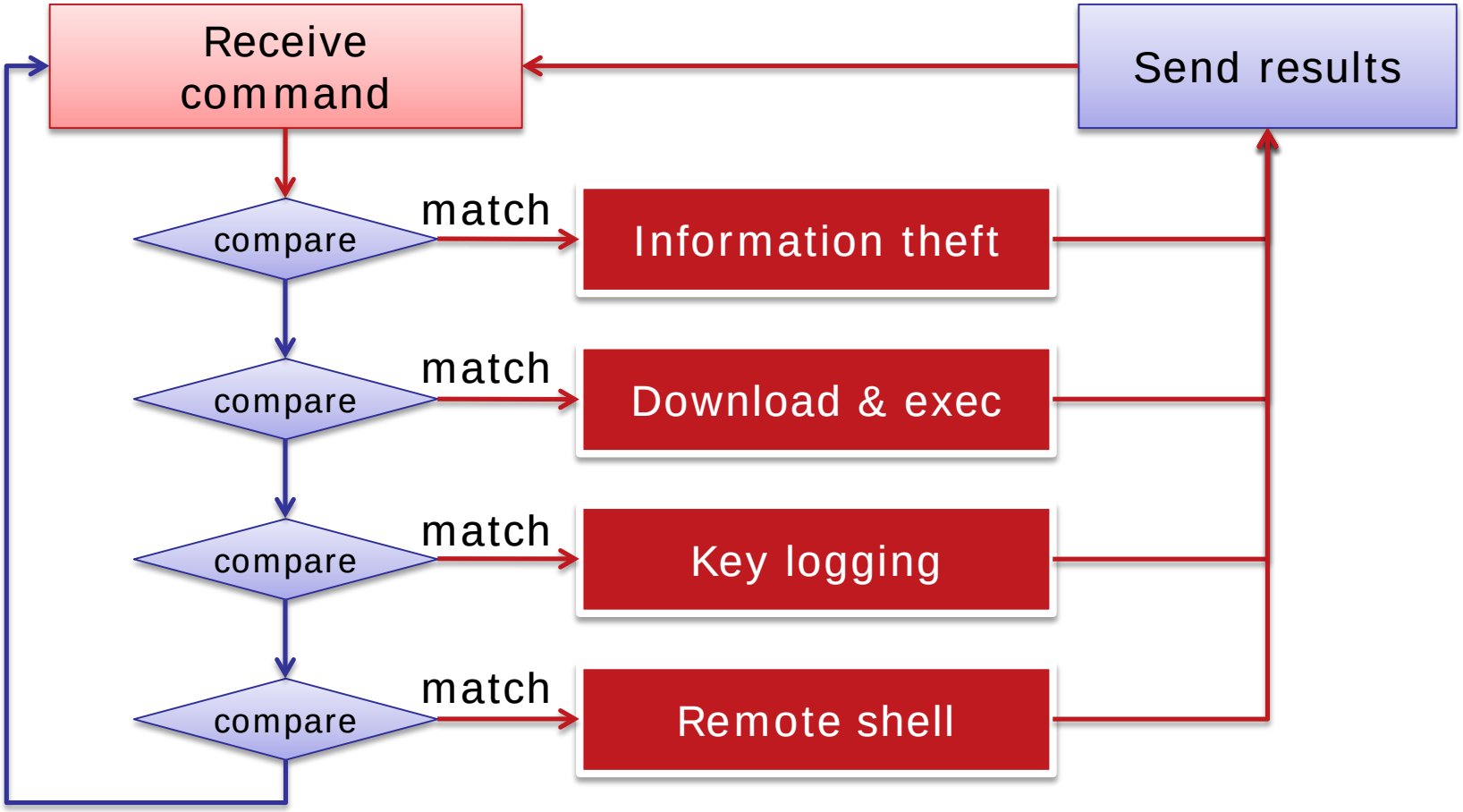
Point 4. Encoding (Obfuscation)

- e.g. HTTP packet obfuscation
  - —Data encoded using "xor" or "ror/rol" may became non-ASCII
  - —Combination with base64 encoding is a common approach



Point 5. Bot command

- Bots are capable to communicate with C&C servers to get commands to work



Exercise 1. Malware Analysis

i. Describe the following points of the target

—Details of each bot command

- **"upload_": Download file from arbitrary URL**
- **"uploadexec_": Download & execute file**
- **"xxx_": Execute arbitrary shell command (Remote shell)**
- **"xxxx_": Upload specific file to C&C server**

—Decode method

- Try to decode malware_sample1_data.bin
- **Wide char -> Multi byte char -> xor 0x53**

ii. Make your IDB

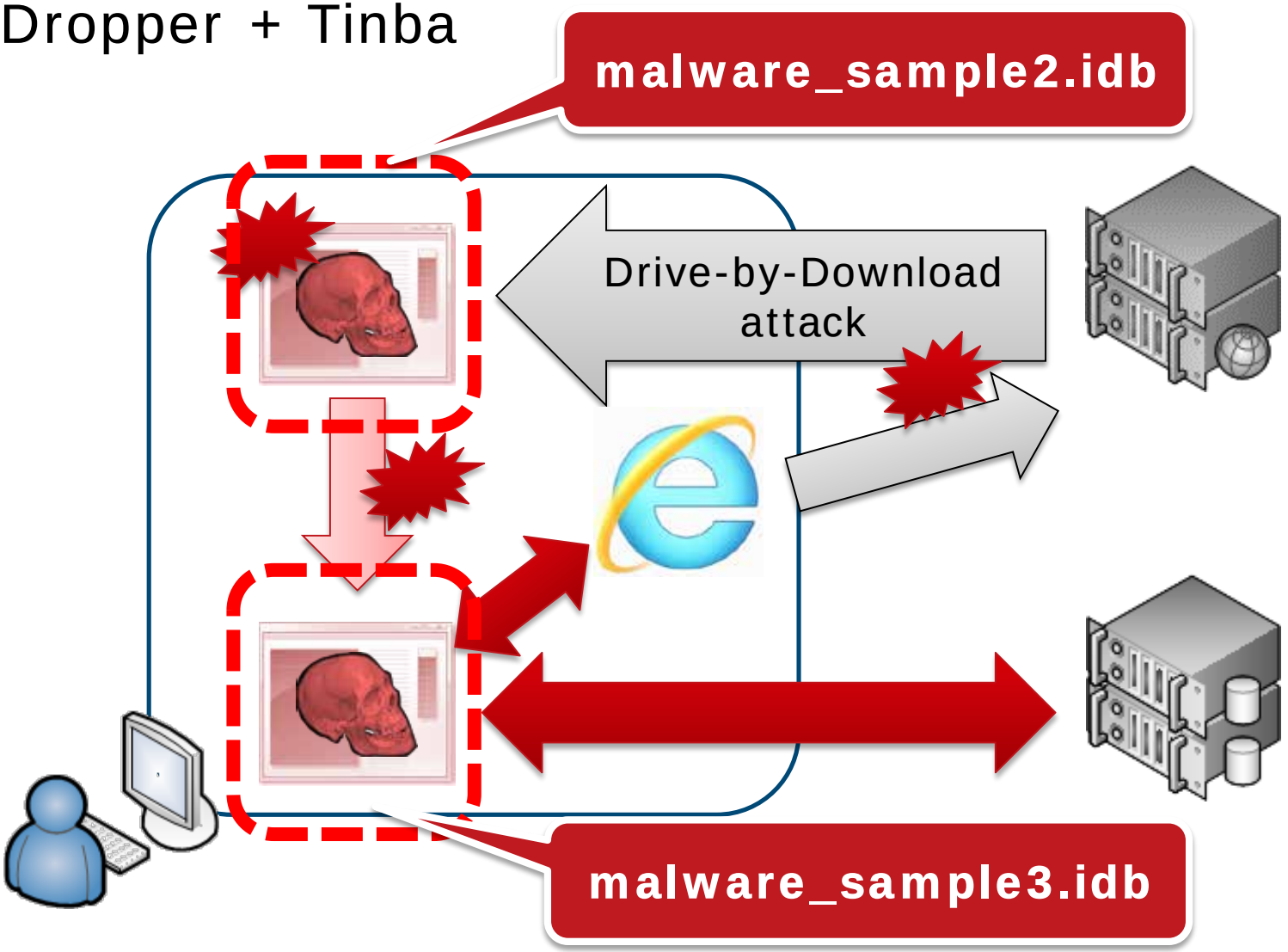
—Fill in information that you analyzed

Malware Analysis

LET'S ANALYZE BANKING TROJAN

Analysis Target

- Dropper + Tinba



Exercise 2. Malware Analysis

i. Analyze position independent data addressing in "malware_sample3.idb"

ii. Analyze "malware_sample_clean.idb" and describe the following points of the target
   - How to avoid anti runtime analysis technique
   - Installation flow
   - Target web browser

iii. Make your IDB
   - Fill in the information that you analyzed

Point 1. Dropping Files

- Create another file
  - —Dropped files usually contains the main function for the attack
- 2 common methods

Download from the server

- Downloader

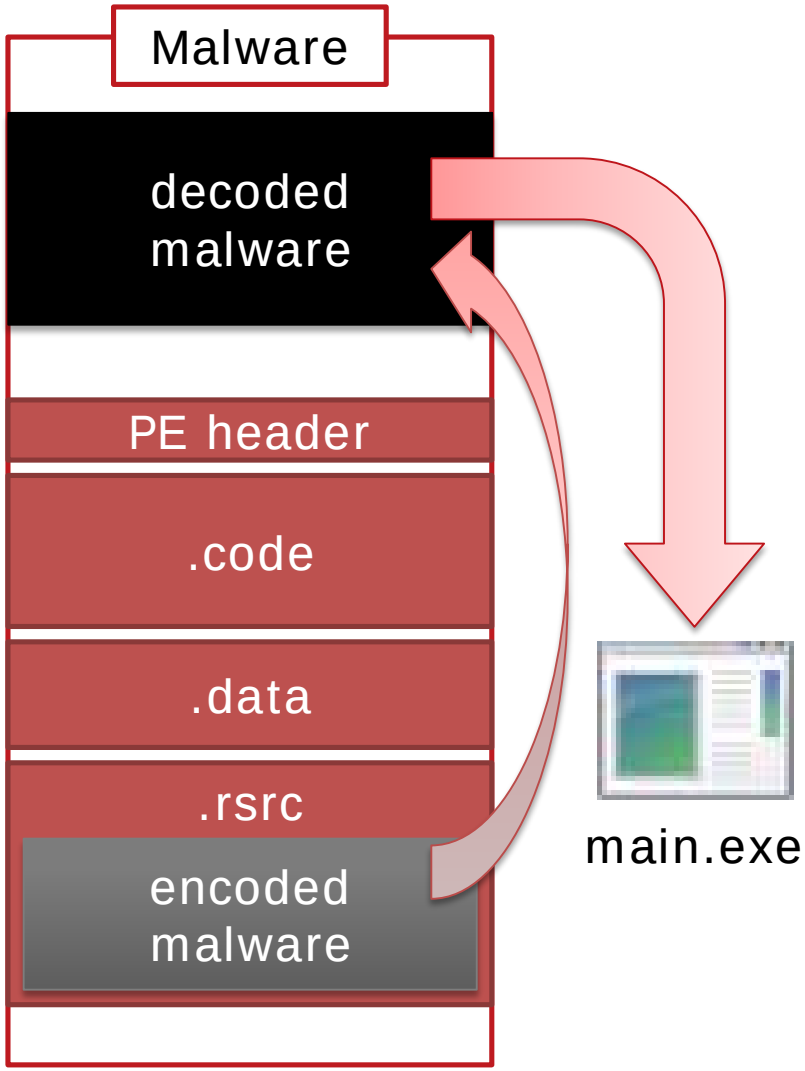
Store drop files in programs

- Data / resource / overlay
  - Usually encoded

Point 1. Dropping Files

- Dropping file from resource

1. Find and load the encoded data from resources
   - n FindResource
   - n LoadResource
   - n SizeofResource
   - n LockResource
2. Decode
   - n HeapAlloc
   - n RtlDecompressBuffer
3. Write decoded data to the file
   - n CreateFile
   - n WriteFile
   - n CloseHandle



Point 2. Position Independent Data Addressing

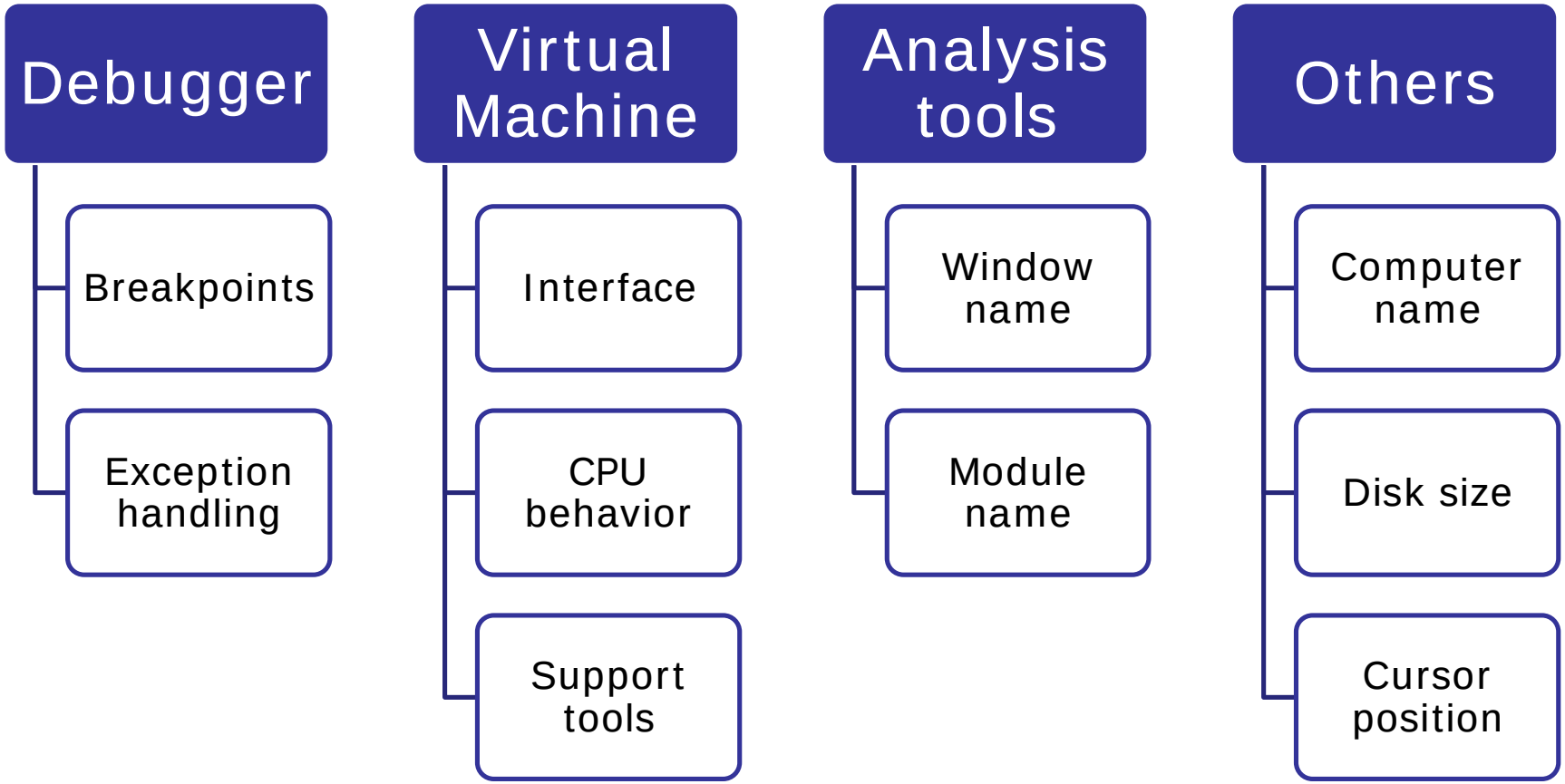
- Push strings using CALL instruction

```
004012C7 00C FF D0                                  call    eax
004012C9 00C 64 8B 35 30 00 00 00                   mov     esi, large fs:30h
004012D0 00C 8B 76 0C                               mov     esi, [esi+0Ch]
004012D3 00C 8B 76 1C                               mov     esi, [esi+1Ch]
004012D6
004012D6                            loc_4012D6:                             ; CODE XREF:
004012D6 00C 8B 4E 08                               mov     ecx, [esi+8]
004012D9 0                                          mov     edi, [esi+20h]
004012DC                                            mov     esi, [esi]
004012DE                                            cmp     dword ptr [edi+0Ch], 320033h
004012E5                                            jnz     short loc_4012D6
004012E7                                            push    ecx
004012E8                                            lea     eax, sub_401219[ebx]
004012EE 010 FF D0                                  call    eax
004012F0 010 E8 06 00 00 00                         call    loc_4012FB
004012F0                            ; ---------------------------------------------------------------
004012F5 010 6E 74 64 6C 6C 00      aNtdll          db 'ntdll',0
004012FB                            ; ---------------------------------------------------------------
004012FB
004012FB                            loc_4012FB:                             ; CODE XREF:
004012FB 010 FF 93 51 10 40 00                      call    ds:dword_401051[ebx]
00401301 010 50                                     push    eax
00401302 014 8D 83 19 12 40 00                      lea     eax, sub_401219[ebx]
00401308 014 FF D0                                  call    eax
0040130A 014 E8 07 00 00 00                         call    loc_401316
0040130F 014 77 73                                  ja      short loc_401384
```

Push address of "ntdll" & jump to next instruction

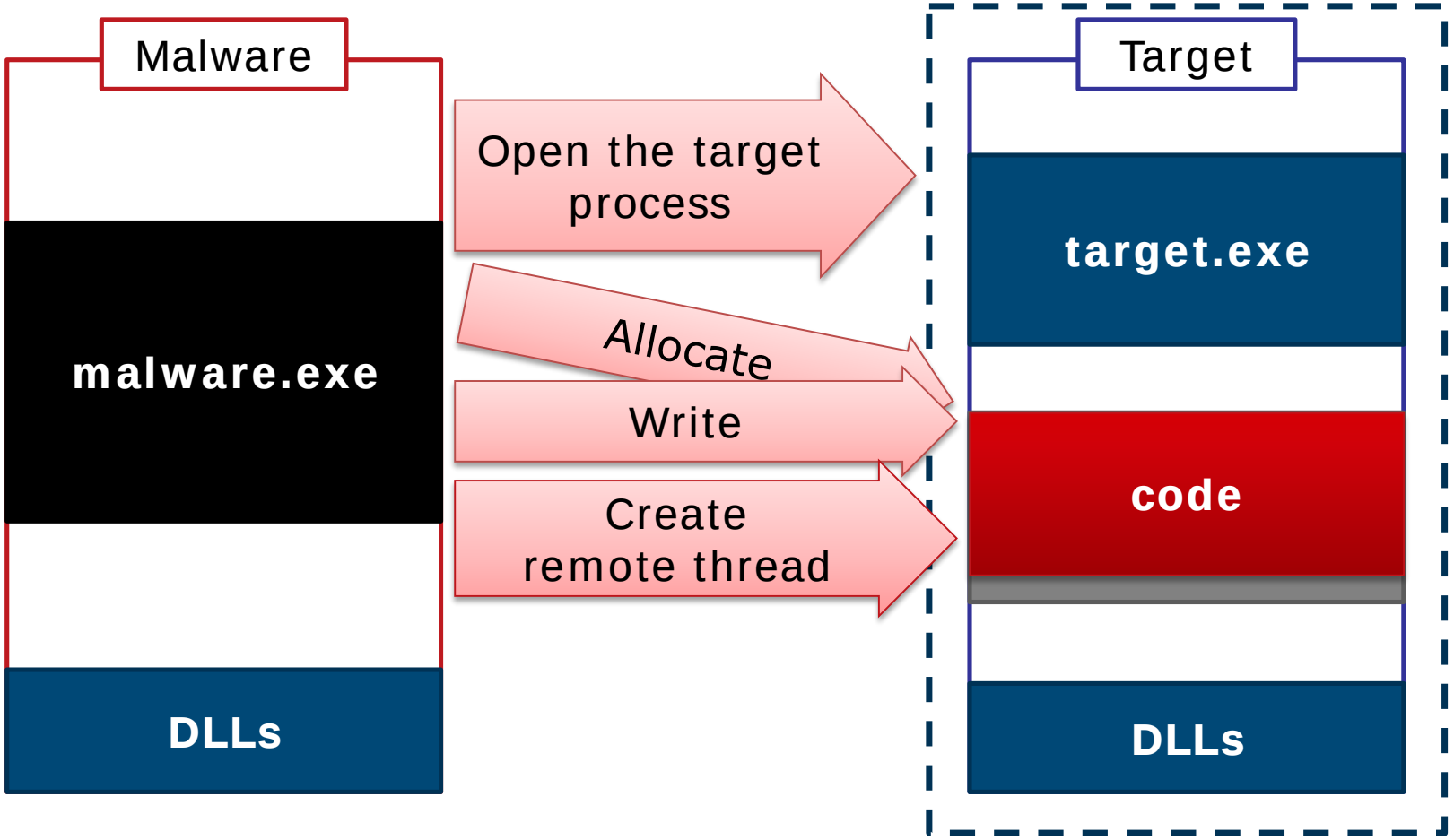
Point 3. Anti Runtime Analysis

- Some types of malware are clever enough to detect analysis activity
  - —To avoid analysis by malware analysts



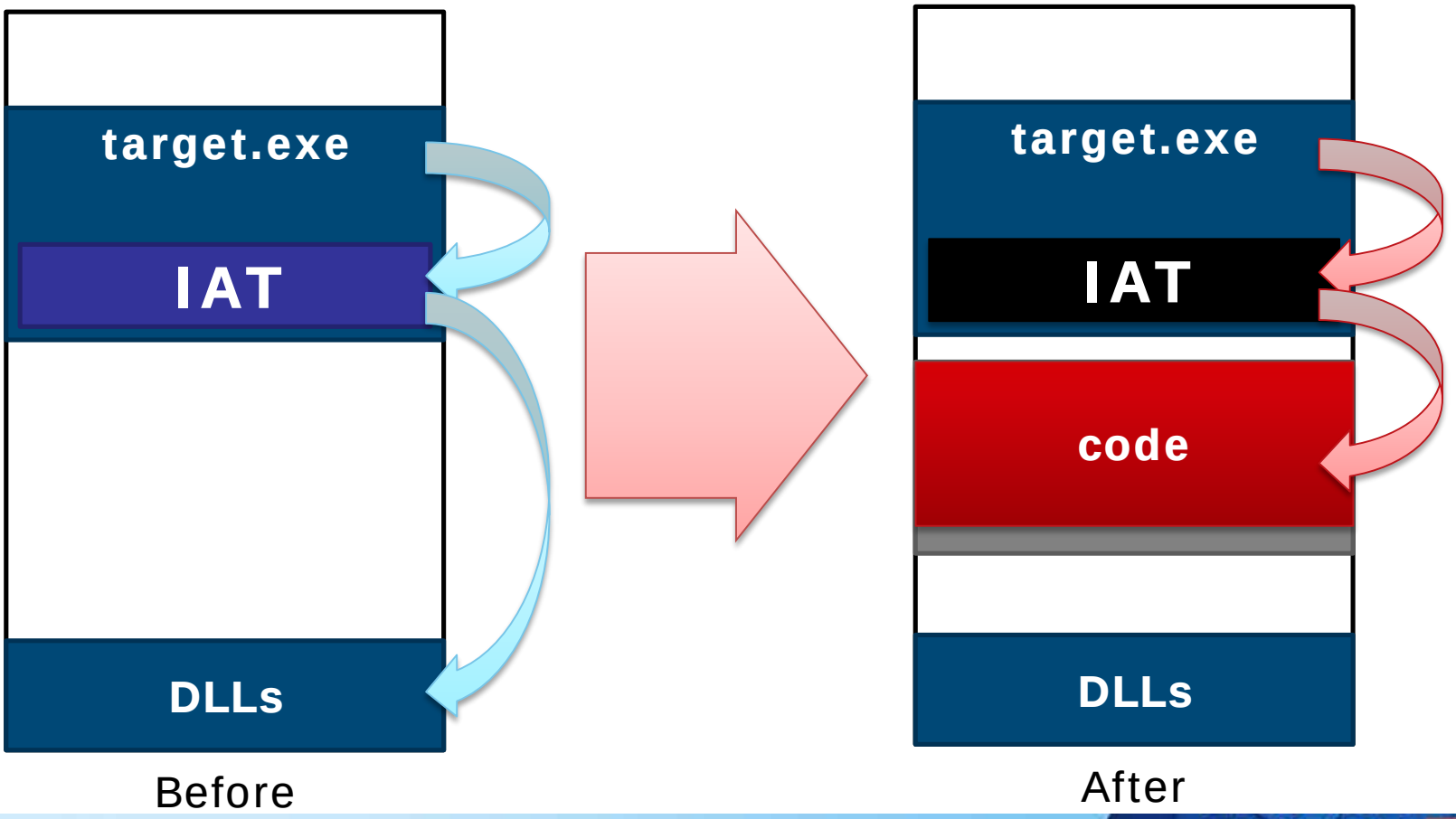
Point 4. Code Injection

- Method to execute arbitrary code in another process



Point 5. API Hooking

- Method to execute arbitrary code when API is called
  - —Logging/Modifying parameters passed to APIs



Exercise 2. Malware Analysis

i. Analyze position independent data addressing in "malware_sample3.idb"

ii. Analyze "malware_sample_clean.idb" and describe the following points of the target
   - —How to avoid anti runtime analysis technique
     - **Mouse cursor checking, Disk cylinder checking**
   - —Installation flow
     - **See "aa_install_as_speechengines" function**
   - —Target web browser
     - **Internet Explorer, Firefox, Chrome, Maxthon**

iii. Make your IDB
   - —Fill in the information that you analyzed

Bonus: Shellcode Analysis

BASIC KNOWLEDGE

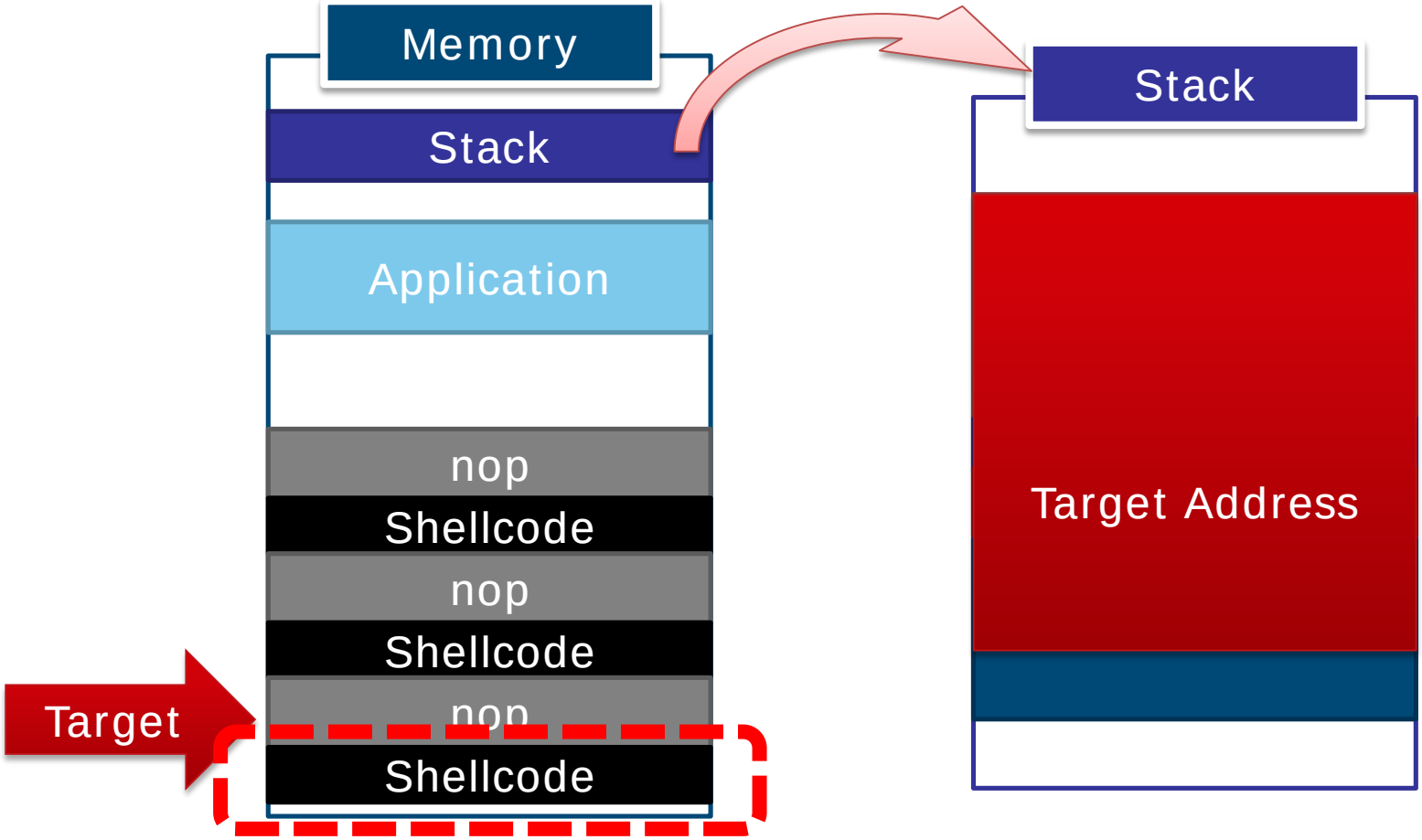
(recap) Exploiting Vulnerability

- Buffer overflow, etc.
- Take control and execute arbitrary code

- Shellcode for malware execution
- Malware

What Shellcode is

- Code snippet that is executed after exploiting
- e.g. Stack based buffer overflow + Heap spray

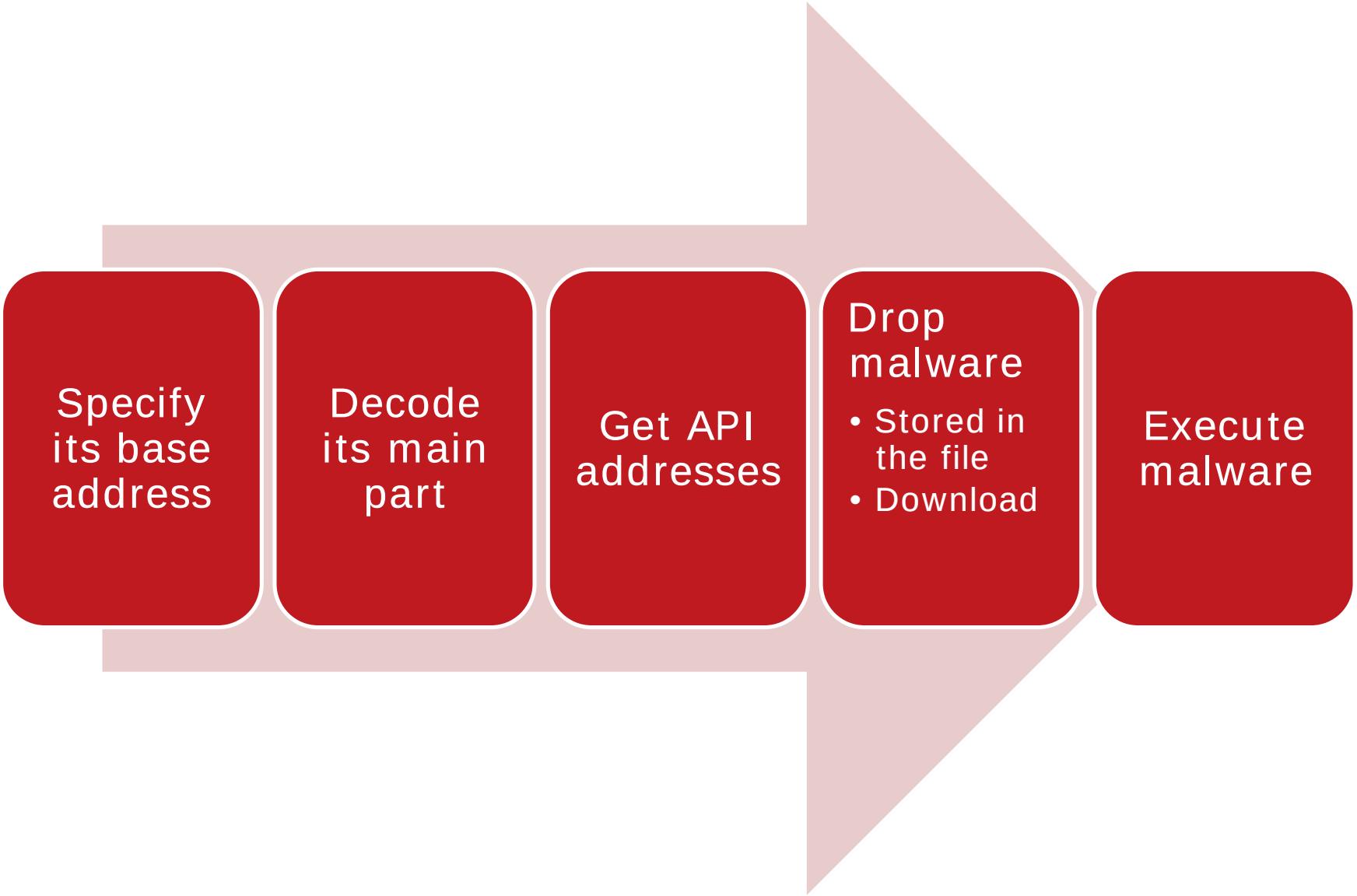


Comparison With Executable File

	Executable file	Shellcode
Format	PE file format (header, code, data, etc.)	Code only
Load address	Specified at PE header	N/A
API address	PE loader will resolve API address	N/A

Shellcode has some routines to retrieve these addresses

Basic Process of Shellcode



TIB & PEB

Thread Information Block (TIB)

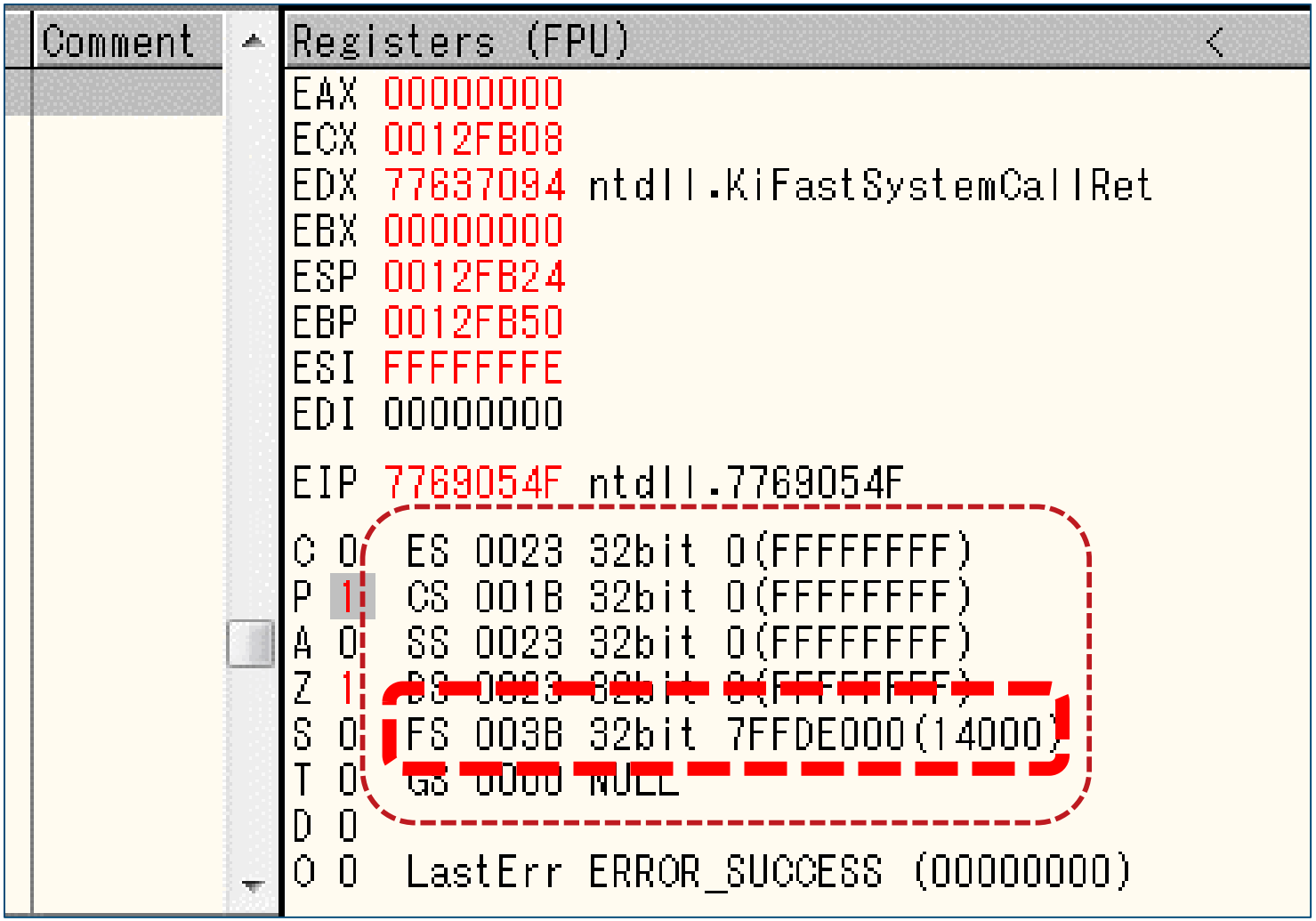
- Also called "Thread Environment Block (TEB)"
- Contains thread related information
  - Thread context, PEB, etc.

Process Environment Block (PEB)

- Contains process related information
  - PID, Loaded modules, etc.

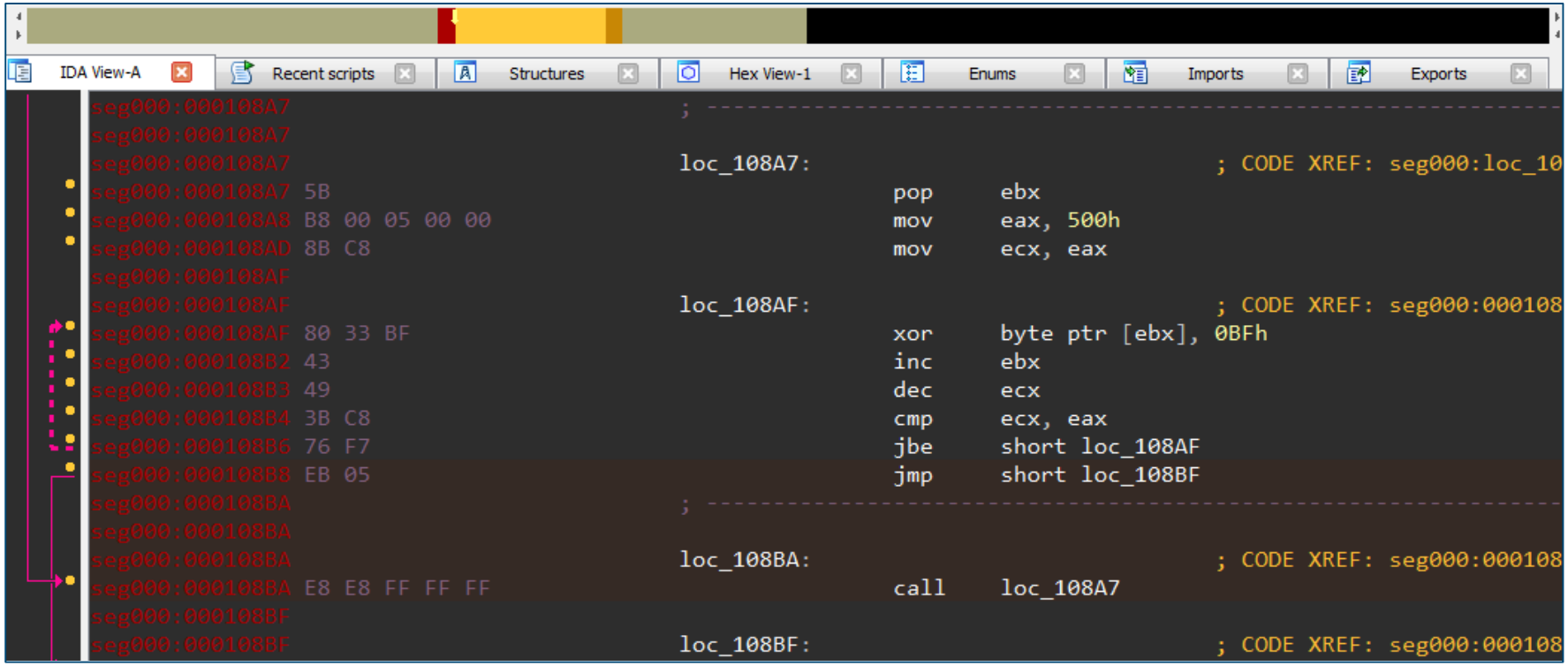
TIB in Segment Register

- FS register points to TIB



Loading to IDA

- Load as a 32bit code
- Recommendation
  - —Change loading offset to 0x00010000 to avoid analysis failure (in some cases)

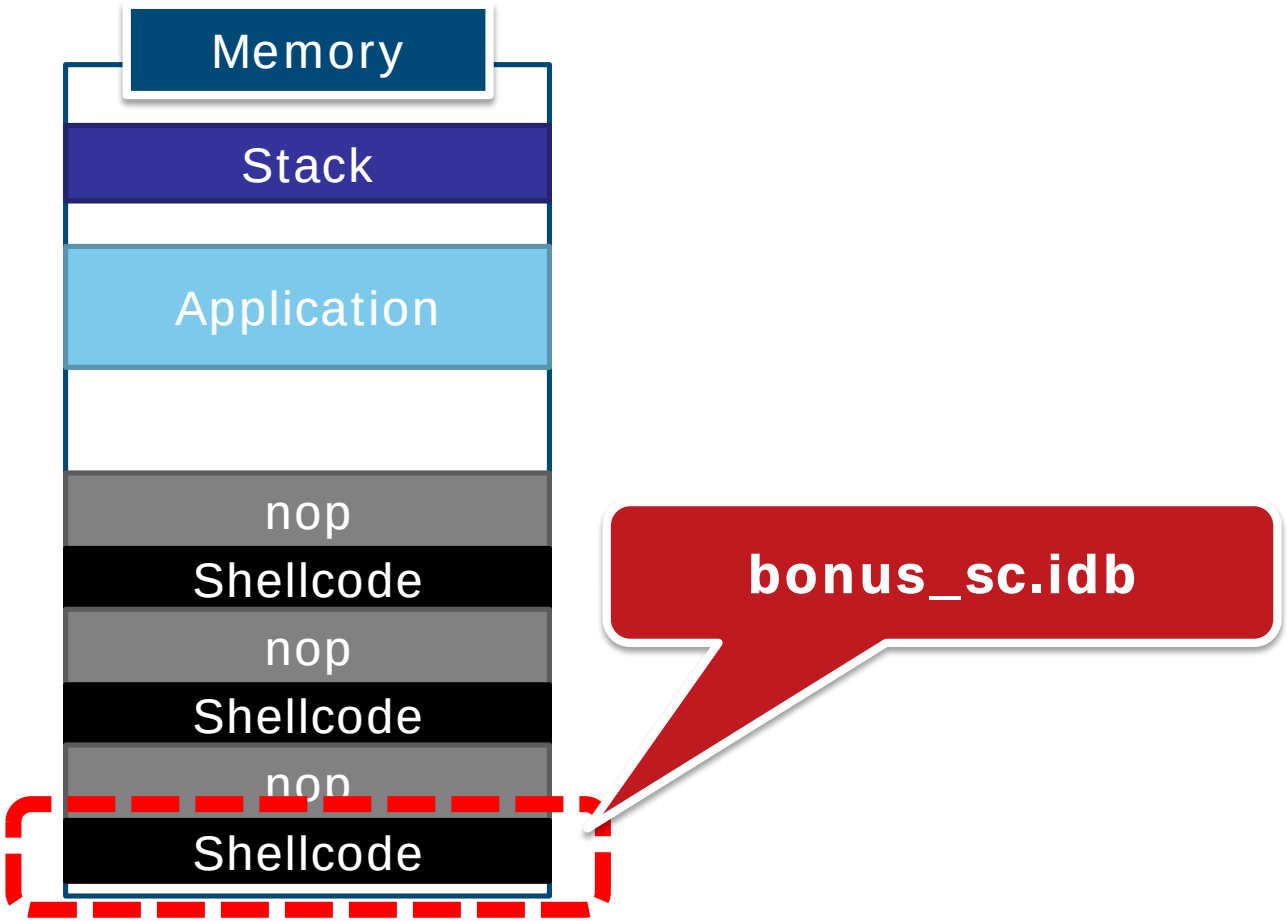


Shellcode Analysis

LET'S ANALYZE

Analysis Target

- Shellcode cropped from memory dump



Point 1. Getting Base Address

- To calculate relative address

Jump to next instruction

- call $+5

push return address

Get return address from stack

- pop ebp

Calculate base address

- sub ebp, 0Dh

Point 2. GetProcAddress

- Step 1: getting base address of kernel32.dll

PEB

- fs:[eax+30h]

Ldr

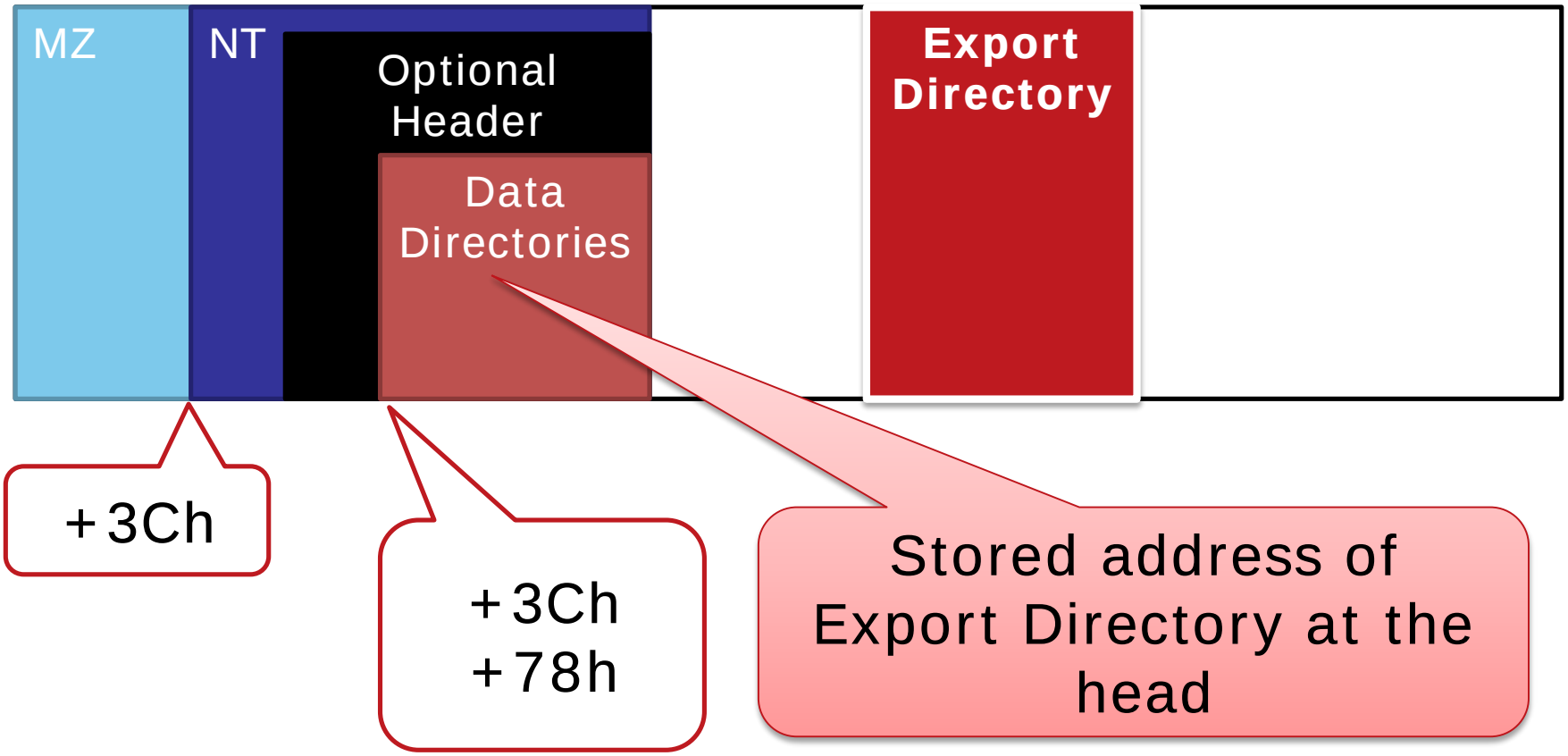
- [eax + 0Ch]

InInitializationOrderModuleList

- [eax+1Ch]

Point 2. GetProcAddress

- Step 2: parsing DLL file to get API address



Point 2. GetProcAddress

- 2 methods to obtain API addresses

Get all API address manually

- Parse DLL file every time
  - Compare export function name with API to use

Use GetProcAddress

- Use GetProcAddress after getting address of GetProcAddress

FYI. API Hashing

- Recent shellcode use hash value of API name for anti-virus/analysis

```
push    ebx
push    esi
push    edi
push    0D5786h          ; kernel32.dll!LoadLibraryA
push    0D4E88h
call    aa_get_proc_address_from_hash
mov     [ebp+var_4], eax
push    348BFAh          ; kernel32.dll!GetProcAddress
push    0D4E88h
call    aa_get_proc_address_from_hash
mov     [ebp+var_8], eax
jmp     loc_100013F
```

See: http://blog.fireeye.com/files/win32_api_hash_table-2.html

Discussion

Questions?