VX-CON

Anti-Debugging and debugging for windows

Kelvin Chan

About me

- Undergraduate in HKBU, major in Computer Science
- Engage in Windows kernel research about 5-6year

Overview

Explanation of debugging

- Background knowledge
- What is actually “Debug” ?
- What is “assembly language”?
- How the program birth and runs?
- Mid-Point

Anti-debugging

- Value of Anti-debugging
- Anti-debugging trick
- Inline hook
- RING3 level API hooking
- SSDT Hook
- RING3 level data structure of OS
- RING0 level data structure of OS

Demo for bypass commercial anti-debugging tools (NProtect)

What is “Debug”?

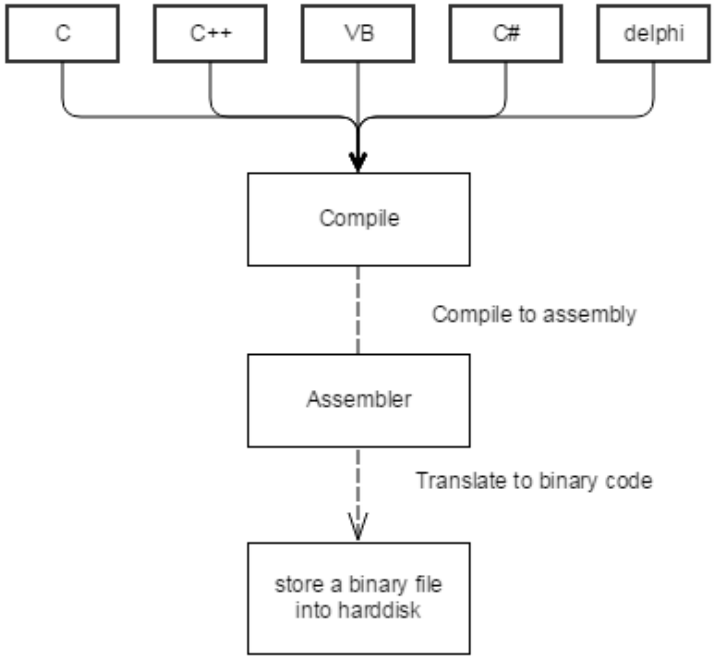
Debug

- **For aspect of development**, can be defined as find an bug that we can see.
- **For aspect of maintain**, can be defined as find bug that we can’t see.
- And how about that for aspect of you and me?
- **For me**, “debug” is a only way for implementation of hack related tools
- Such as, anti-virus, online-game cheating

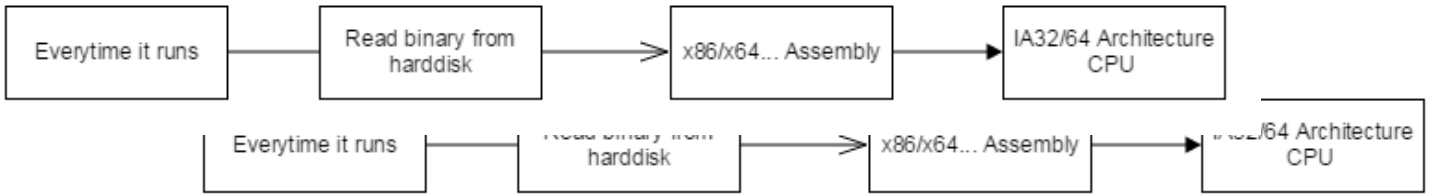
What is assembly language

- Assembly language is a machine language
- Every this Instruction actually is a binary code for CPU execute
- Such as:
  - 0x90 == NOP instruction
  - FF 25 00 00 00 00 == JMP 00000000
  - E9/E8 xx xx xx xx is CALL/JMP to a offset relative to instruction address
  - …. So on
- All of instruction has a rule for mapping Hex.(binary)number to instruction, which defined in intel ia32/64 structure

1.


C
C++
VB
C#
delphi
Compile
Compile to assembly
Assembler
Translate to binary code
store a binary file into harddisk

2.


Everytime it runs
Read binary from harddisk
x86/x64... Assembly
IA32/64 Architecture CPU
Everytime it runs
harddisk
x86/x64... Assembly
CPU

Mid-Point

- We know all program must finally compile as binary code, run as a binary code
- So for the non-open source language / Operating System , we use that logic to make the binary convert back to assembly language.
- Finally we can know the targeted programming logic, such as, some procedure CALL.
- That i simply explained what software hacker / researcher did.

- After that, the non-stop war is started --> birth of anti-debugging

Value of Anti-debugging

As we all know what is the debugging now, and we can realized that how it is actually threatening the benefits of our programmers / computing related firm.

Set a target: Make a debugger cannot be run normally is ONLY things we needed now

The debugger tools :

- Ollydbg
- IDA Pro (mainly for static disassembly,but not oly)
- Windbg (kernel level debugging)
- x64/x86dbg

Current Anti-debug Firms

- NProtect gameguard
- TenProtect
- HackShield
- X-trap

Anti-debugging tricks

- Any method makes a debugger process crashes, which means we are success.

There are two aspect we can consider to achieve this target :

- some method in OS can be directly exploited (some debugging flags)
- some method in OS can be indirectly exploited(something not related debugging)

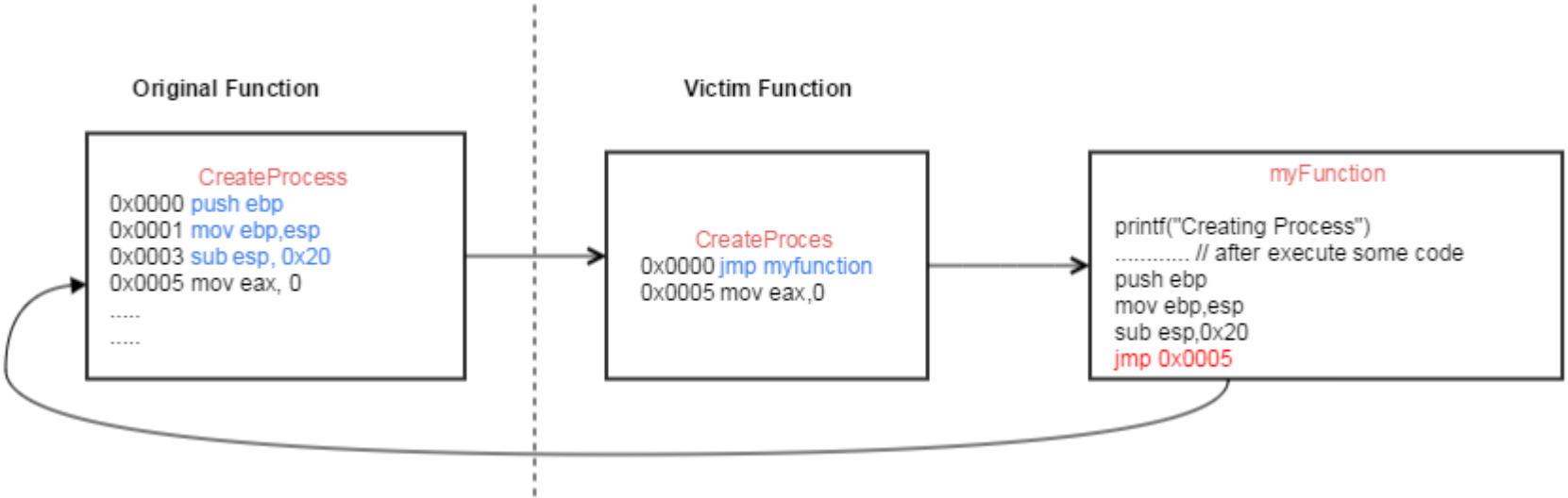
For example:

- OS level API CALL Hook
- OS level Data-Structure monitor and detection

Inline hook

Inline hook, implementation is based on instruction modification

It can be done by any platform, we can any apply the concept of this hook into any instruction structure



RING3 level API hooking

When the debugger is trying to start-up a debuggee, it must use different kind of method for creating the process. Hacker / Anti-debug manufacturer can easily use the inline hook for hooking the following function

For RING 3 example:

- Kernel32 ! CreateProcess
- Kernel32 ! CreateProcessW
- Kernel32 ! CreateProcessA
- Kernel32 ! CreateProcessAsUser
- Kernel32 ! CreateProcessInternalW
- Ntdll ! NtCreateProcess

RING0 level API Hooking

For RING 0 example:

- nt!NtCreateProcess
- nt!NtCreateProcessEx
- nt!NtCreateUserProcess
- … so on

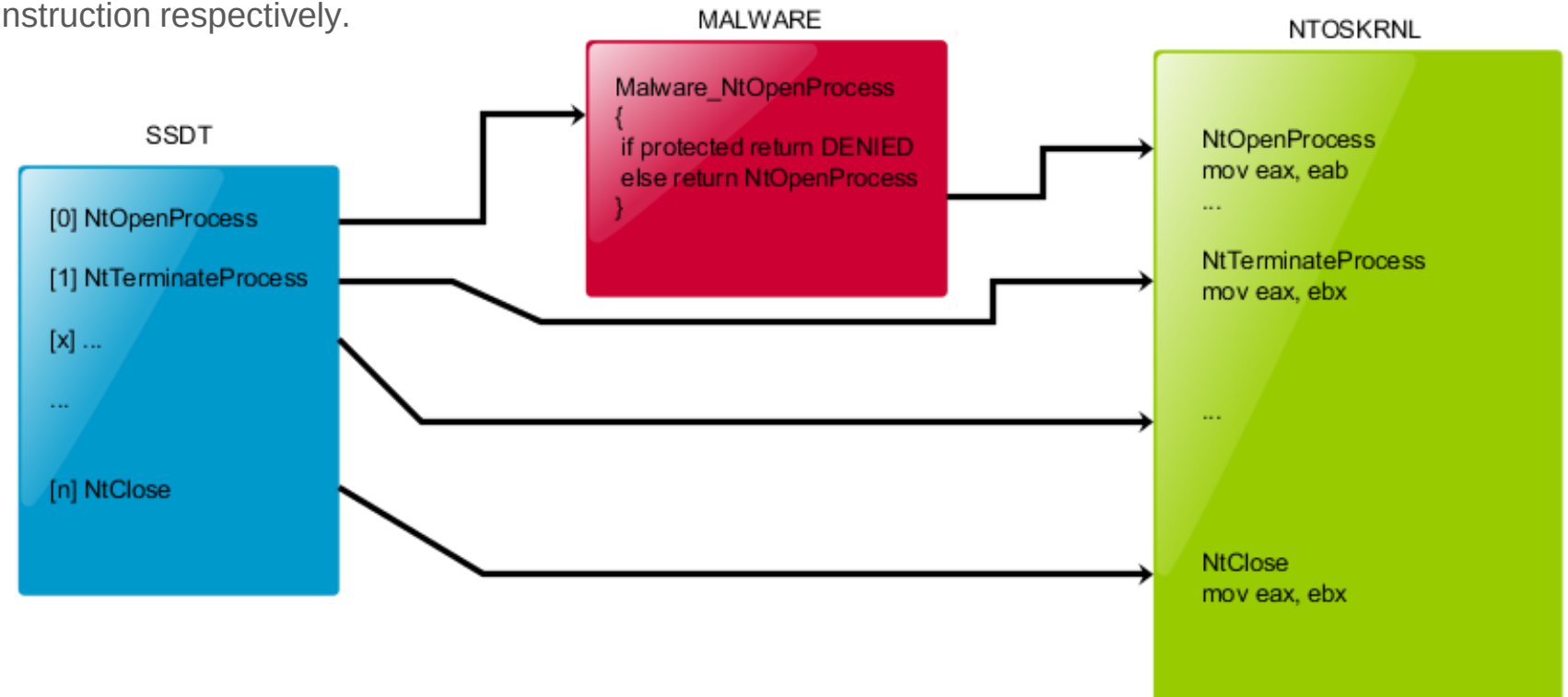
Nt* function pointer stored in the table called ***System Service Descriptor Table (SSDT)***

* We can easily modify the table entry for intercept purpose

p.s. As a hints for who wants to be a blackhat on windows: although SSDT hook is weak for real confrontation, but for x64 system, which is always work for attacking legal company, because, they always can’t modify kernel

SSDT Hooking

System call for Windows x86 and x64 will be performed by SYSENTER (0F 34) and SYSCALL(0F 05) instruction respectively.



Ring3 data-structure of OS

Related flags of Debugging

For example :

- **Process Environment Block(PEB)**
    - Use for obtaining process’s related information
    - Has a flag indicated the process is it being debug (called **BeingDebugged**)
    - Some of Anti-Debugging has a thread infinitely check this flag is it being active
- **Thread Environment Block (TEB)**
    - Use of obtaining thread’s related information

RING0 Data Structure of OS

Kernel object is a very important for windows platform

- It is not a real object , but it is a conceptually object-oriented
- In windows every instance encapsulated as an "object” by struct of C language or even assembly language.
    - Instance, such as, Process, Thread, Section, File, etc.
    - Kernel object descripting the instance message such as, process id, process creation time, process current status, how many page mapped, virtual address description, etc.

Another structure is ***descriptor***

- Descriptor always descripted how is another data stored, such as, elements of **Interrupt Descriptor Table(IDT)**, each entry is a descriptor for descripting each interruption of CPU.

Direct Kernel Object Manipulation(DKOM)

An operating system always provide a struct for maintain different instance in OS, for maintain them easily within kernel internal and CPU, implied there is a war of hacker and anti-debugging engineer.

For example:

- **EPROCESS**
- **KPROCESS**
- **ETHREAD**
- **KPROCESS**

EPROCESS

EPROCESS is a type of *windows kernel object*, which is responsible for managing different process, it is a very big structure, unsurprisingly , which will be exploited by many hacker.

EPROCESS have different member can be exploited for hacking usage.

For example:

EPROCESS-> DebugPort; // which is extremely important, because it stored the correspodning address of debug_object, we only need to write a zero into target debugobject, other debugger will not receive any debug message from the process

EPROCESS->VadRoot ; // Root of AVL tree, node that Stored the virtualAddress descriptor, such as, memory mapping status, we can take it off from the tree, achieving memory hiding (such as, some section we want to be hidden)

Kernel level CALLBACK

Kernel level CALLBACK which is a interfaces that provide by Windows.
After the CALLBACK is set, the specific action of whole OS will be under our monitor, even control.

For example:

- PsSetCreateProcessNotifyRoutine
- PsSetCreateThreadNotifyRoutine
- PsSetLoadImageNotifyRoutine (callback from MmMapViewOfFile -> MiMapViewOfImageSection for create memory mapping file , such as , Load DLL)

Background knowledge

As a better reverse engineer or so-called hacker in Windows, we should take over these knowledge :

- At least, familiar with C and x86/x64 assembly
- Familiar with Windows RING3 level knowledge
- Familiar with Windows RING0 level knowledge and debugging mechanism(how OS supported debugging feature)
- Knowledge at least know 70% or above of process, thread and memory. (since these is the basic component of OS)

Demo for bypassing NProtect anti-debugging

Go

What is the challenge of bypassing NProtect?

- Global DLL injection, up to 20 function hooked in each process
- Hide process
- Hide thread from debugger
- Debug_port non stop detect is it zero
- Debug_mask non stop detect is it zero
- Up to 20 hooking in kernel module(very important)

Thanks

E-mail / Facebook: kelvin1272002@hotmail.com