API Hooking

[개념]

> *리버싱에서 후킹은 정보를 가로채고, 실행 흐름을 변경하고, 원래와는 다른 기능을 제공하게 하는 기술*

1. Process

- 디스어셈블러/디버거를 이용하여 프로그램 구조와 동작원리를 파악
- 버그 수정 또는 기능 개선에 필요한 Hook Code 개발
- 실행 파일과 프로세스 메모리를 자유롭게 조작하여 Hook Code 설치

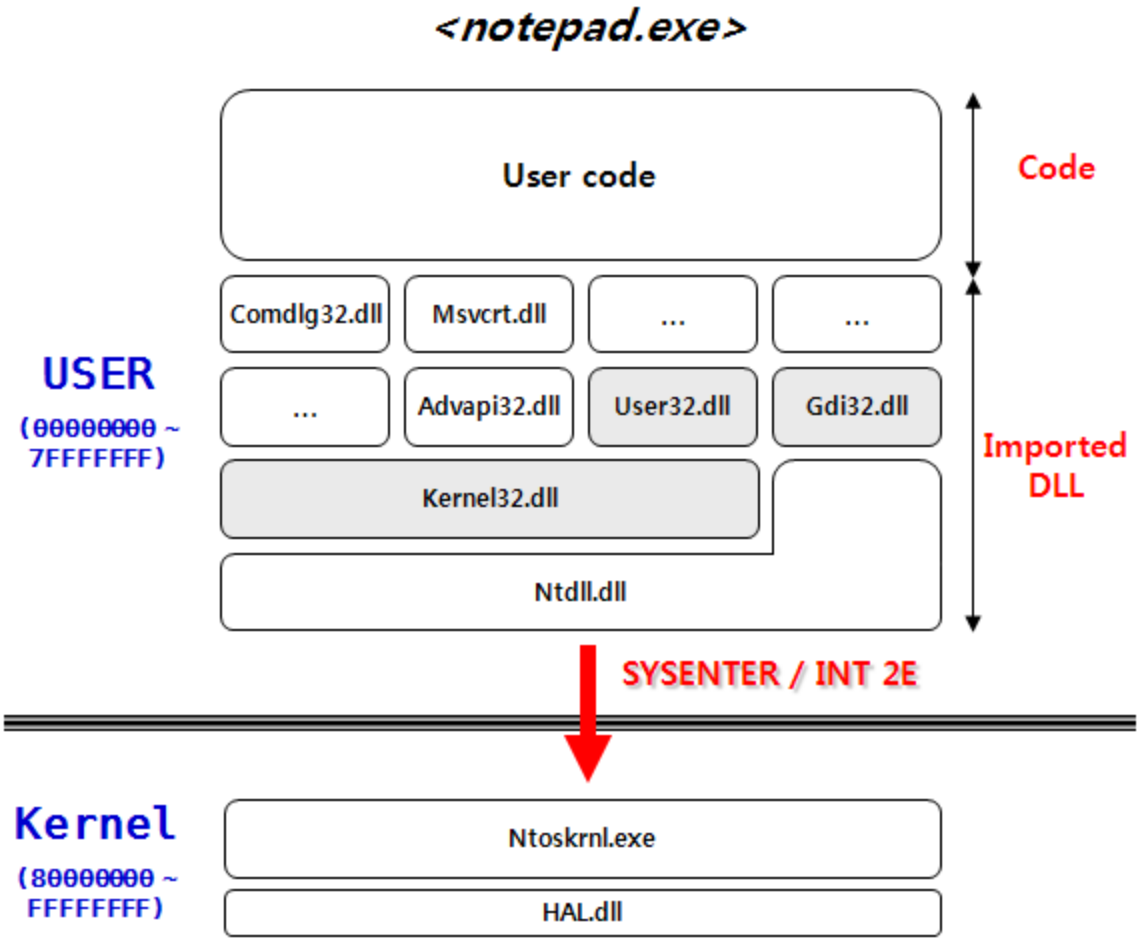
2. 종류

- Kernel Mode
  - API Hooking
- User Mode
  - Message Hooking

[API (Application Programming Interface)]

> *Windows OS에서는 사용자 application이 system resource(memory, file network, video, sound, etc)을 사용하고 싶을 때 직접 할 수 있는 장법이 없다.*
> *왜냐하면 그것들은 OS가 직접 관리하며, 여러가지 이유(안정성, 보안, 효율 등)로 사용자 application의 직접적인 접근을 막아놓았기 때문이다.*
> *즉, System Kernel에게 요청해야 하며, 요청하는 벙법은 API를 활용하면 되고, 각 OS에 API는 각 OS 제작사에서 제공한다.*

1. Application Structure



실제 어플리케이션 코드를 실행 시키기 위해 많은 시스템 라이브러리(DLL)들이 로딩된다.

> *모든 프로세스에는 기본적으로 kernel32.dll이 로딩되며, kernel32.dll은 ntdll.dll을 로딩한다.*

- ntdll 역활
  - 유저 모드 application의 code에서 발생하는 시스템 자원에 대한 접근을 커널 모드에게 요청하는 것이다.

2. notepad.exe에서 c:\abc.txt 파일을 열고자 한다.

- 코드에서는 msvcrt!fopen() API를 호출한다.
- API 호출 흐름을 아래와 같다.
  - 일반적인 시스템 자원을 사용하는 API는 kernel32.dll과 ntdll.dll을 타고 가다가 결국 SYSETER 명령을 통해 커널 모드로 진입하게 된다.

```
msvcrt!fopen()
  kernel32!CreateFileW()
    ntdll!ZwCreateFile()
      ntdll!KiFastSystemCall()
        SYSENTER                    // Intel IA-32 Op Code
          => 커널 모드 진입
```

Tip. DLL Loading (X) → DLL Mapping (0)

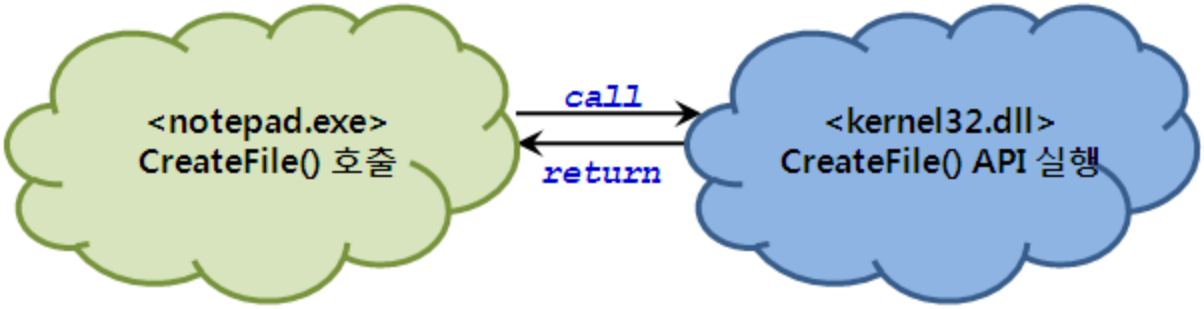
- windows 운영체제는 DLL을 최초 한번만 메모리 적재(loading)하고, 그 이후부터는 프로세스에게 매핑(mapping) 시켜주는 메커니즘을 사용합니다.

[API Hooking]

> *Win32 API 호출을 중간에서 가로채서 제어권을 얻어내는 것이다.*

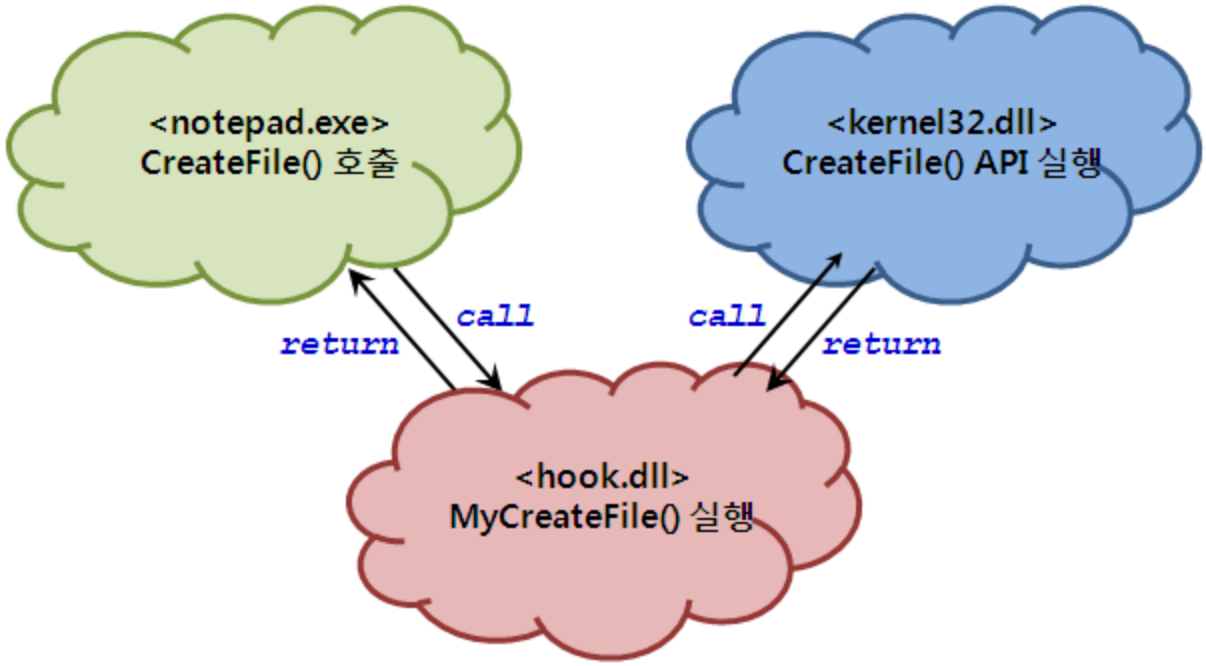
1. API 호출

1) 정상적인 호출



- Code 영역 주소에서 CreateFile() API 호출
  - CreateFile() API는 kernel32.dll에서 서비스(export)하므로 kernel32.dll영역의 CreateFile() API가 실행되고 정상적으로 반환한다.

2) 비 정상적인 호출; kernel32!CreateFile() API가 후킹된 경우



- 사용자가 DLL Injection 기술로 hook.dll을 프로세스 메모리 공간에 침투
  - kernel32!CreateFile()를 hook!MyCreateFile() 로 후킹
- 이제부터 해당 프로세스에서 CreateFile() API 호출시 hook!MyCreateFile()이 호출된다.
  - 후킹 함수와 원본 함수의 호출 순서는 경우에 따라 달라진다.
    - 입력된 파라미터 조작 : 후킹 함수 호출 → 원본 함수 호출
    - 반환값 조작 : 원본 함수 호출 → 후킹 함수 호출

[API Hooking Tech map]

<table>
<tr><th>Method</th><th>Object
(what)</th><th>Location
(where)</th><th colspan="2">Technique
(how)</th><th>API</th></tr>
<tr><td>static</td><td>File</td><td rowspan="5">1) IAT
2) Code
3) EAT</td><td colspan="2">X</td><td>X</td></tr>
<tr><td rowspan="4">dynamic</td><td rowspan="4">Process
Memory

00000000
~
7FFFFFFF</td><td colspan="2">A) Debug
(Interactive)</td><td>DebugActiveProcess
GetThreadContext
SetThreadContext</td></tr>
<tr><td rowspan="3">B) Injection
(stand
alone)</td><td>B-1)
Independant
Code</td><td>CreateRemoteThread</td></tr>
<tr><td rowspan="2">B-2) DLL file</td><td>Registry (AppInit_DLLs)
BHO (IE only)</td></tr>
<tr><td>SetWindowsHookEx
CreateRemoteThread</td></tr>
</table>

1. Method - Object (what)

> *API 후킹 방식(Method)에 대한 대 분류*
> *Method는 작업 대상(Object)에 따라서 Static과 Dynamic 방식으로 나뉜다.*

- Static Method
  - 작업 대상 : File
  - 프로그램 실행 전 후킹
  - 최초 한번만 후킹
  - 특수한 상황에서 사용됨
  - Unhook을 위해 프로그램 종료해야 함
- Dynamic Method
  - 작업 대상 : Process Memory
  - 프로그램 실행 후 후킹
  - 실행 될 때마다 후킹
  - 일반적인 후킹 방법
  - 프로그램 실행 중 Unhook 가능 (유연성 제공)

2. Location (Where)

> *작업 대상의 어느 부분을 공략(조작)해야 하는지에 대한 내용*

1) IAT (Import Address Table)

- IAT에 있는 API 주소를 후킹 함수 주소로 변경하는 방법
- 장/단점
  - 장점 : 가장 단순, 구현 방법이 쉬움

        - 단점 : IAT에 없는데 프로그램에서 사용되는 API 들에 대해서는 후킹 불가
            - Ex) DLL을 동적으로 로딩해서 사용하는 API
    - IAT : DLL이 프로그램 시작시 로딩되어 종료시 메모리에서 해제되는 Implicit linking 방법에 대한 Mechanithm 제공

2) Code

> *프로세스 메모리 에 매핑된 시스템 라이브러리(*.dll)에서 API의 실제 주소를 찾아가 코드를 직접 수정해 버리는 방법*
> *참고로 이 방법이 가장 널리 사용되는 방법*

- 여러 가지 옵션
    - 처음 5byte를 JMP XXXX XXXX 명령어로 패치하는 방법
    - 함수 일부를 덮어쓰는 방법
    - 필요한 부분만 일부 변경하는 방법

3) EAT; Export Address Table

> *DLL의 EAT에 기록된 API의 시작 주소를 후킹 함수 주소로 변경하는 방법*

3. Technique (How)

> *후킹 대상 프로세스 메모리에 침투하여 후킹 함수를 설치하는 구체적 기법(Technique)*

1) Debug

> *대상 프로세스를 디버깅하면서 API 후킹을 하는 방법*

- Debugger
    - 후킹을 위하여 사용자가 직접 제작한 프로그램.
        - Debug API 활용 Target Process에 Attach
        - Hooking 함수 설치
    - 디버깅 당하는 프로세스(Debuggee)에 대한 모든 권한(실행 제어, 메모리 액세스 등)을 가지기 때문에 Debuggee 의 프로세스 메모리에 자유롭게 설치 할 수 있다.
        - API 후킹 도중이라도 사용자가 Interactive하게 프로그램의 실행을 멈추고 API 후킹을 추가 / 수정 / 제거 등 작업이 가능하다.

2) Injection

> *해당 프로세스 메모리 영역에 침투하는 기술*
> *Injection 대상에 따라 Code Injection, DLL Injection으로 나뉜다.*

- DLL Injection
    - 대상 프로세스로 하여금 강제로 사용자가 원하는 DLL file을 로딩하게 만드는 기술
    - Injection할 DLL에 미리 후킹 코드와 설치코드를 만들고 DllMain()에서 설치 코드를 호출해주면 Injection되는 순간에 API 후킹이 완료된다.
- Code Injection
    - 기존 DLL Injection보다 좀더 발전된 기술이며, 주로 악성코드에서 많이 사용됩니다.

- 실행 코드와 데이터만 Injection된 상태에서 자신이 필요한 API 주소를 직접 구해서 사용해야 하며, 코드 내의 메모리 주소에 접근할 때 잘못된 주소를 액세스 하지 않도록 매우 주의 해야 한다.

4. API
