/Rooted

new w32 hooking skills

rooted 2010

kd> dt rooted!_ new_w32_hooking_skills_authors

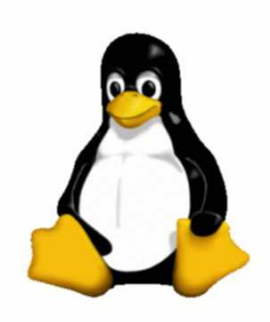
+oxooo David Reguera García: _ROOTED_SPEAKER

SOBRE EL AUTOR …

new w32 hooking skills

rooted 2010

- Investigador en el área de la seguridad informática, destacando sus trabajos de I+D+I e ingeniería inversa en Linux y Windows en la arquitectura x86.



- Tres años de experiencia profesional en el sector, actualmente trabajando en Hispasec Sistemas como Security Researcher.

INDICE

new w32 hooking skills

rooted 2010

PEB HOOKING

new w32 hooking skills

rooted 2010

- El método PEB HOOKING consiste en suplantar una DLL_REAL en memoria por una DLL_FAKE, de forma que todos los módulos de un proceso que usen la DLL_REAL pasen a usar la DLL_FAKE:
  - PEB DLL Hooking Novel method to Hook DLLs, By Deroko, ARTeam-Ezine 2
  - phook - The PEB Hooker, By Juan Carlos Montes Senra (Shearer) y David Reguera García (Dreg), phrack #65

- El Process Environment Block (PEB) es una estructura ubicada en el espacio de usuario, que contiene los datos de entorno del proceso: variables de entorno, lista de módulos cargados, direcciones en memoria del Heap, si el proceso está siendo depurado ...

PEB HOOKING

new w32 hooking skills

rooted 2010

- Para realizar PEB HOOKING es necesario usar el campo LoaderData (Ldr):
  - 0: kd> dt nt!_PEB
    - +0x000 InheritedAddressSpace : UChar
    - +0x001 ReadImageFileExecOptions : UChar
    - +0x002 BeingDebugged : UChar
    - +0x003 SpareBool : UChar
    - +0x004 Mutant : Ptr32 Void
    - +0x008 ImageBaseAddress : Ptr32 Void
    - +0x00c Ldr : Ptr32 _PEB_LDR_DATA

    ....

PEB HOOKING

new w32 hooking skills

rooted 2010

- LoaderData (Ldr) Es una estructura en la que se encuentran algunos datos de los módulos de un proceso. Se trata de una lista doblemente enlazada y que se puede recorrer según tres criterios: orden de carga, orden en memoria y orden de inicialización:
  - o: kd> dt nt!_PEB_LDR_DATA
    - …
    - +oxooc InLoadOrderModuleList : _LIST_ENTRY
    - +ox014 InMemoryOrderModuleList : _LIST_ENTRY
    - +ox01c InInitializationOrderModuleList : _LIST_ENTRY
    - …
- Cada campo flink y blink de LIST_ENTRY son en realidad punteros a LDR_MODULE.

PEB HOOKING

new w32 hooking skills

rooted 2010

- Los datos que vamos a manipular de LDR_MODULE para realizar PEB HOOKING son: BaseAddress: La base del módulo en memoria, EntryPoint : Dirección donde se encuentra la primera instrucción a ejecutar del módulo, SizeOfImage: Tamaño del módulo en memoria:
  - o: kd> dt nt!_LDR_DATA_TABLE_ENTRY
    - +0x000 InLoadOrderLinks : _LIST_ENTRY

    ...
    - +0x018 DllBase : Ptr32 Void
    - +0x01c EntryPoint : Ptr32 Void
    - +0x020 SizeOfImage : Uint4B

    ....

PEB HOOKING

new w32 hooking skills

rooted 2010

- o: kd> !list -t ntdll!_LDR_DATA_TABLE_ENTRY.InLoadOrderLinks.Flink -x "dt ntdll!_LDR_DATA_TABLE_ENTRY " poi($peb + oxc) + oxc

...

- +0x018 DllBase : 0x7c910000
- +0x01c EntryPoint : 0x7c922c48
- +0x020 SizeOfImage : oxb8000
- +0x024 FullDllName : _UNICODE_STRING "C:\WINDOWS\system32\ntdll.dll"

...

- "C:\WINDOWS\system32\kernel32.dll"

...

PEB HOOKING

new w32 hooking skills

rooted 2010

- o: kd> !list -t ntdll!_LDR_DATA_TABLE_ENTRY.InLoadOrderLinks.Flink -x "dt ntdll!_LDR_DATA_TABLE_ENTRY " poi($peb + oxc) + oxc

...

- +0x018 DllBase : 0x7c910000
- +0x01c EntryPoint : 0x7c922c48
- +0x020 SizeOfImage : oxb8000
- +0x024 FullDllName : _UNICODE_STRING "C:\WINDOWS\system32\ntdll.dll"

...

- "C:\WINDOWS\system32\kernel32.dll"

...

PEB HOOKING

new w32 hooking skills

rooted 2010

- **Cuando un proceso, en el que se ha hecho PEB HOOKING, carga un módulo dinámicamente que tenga importaciones de DLL_REAL, automáticamente se cargará su IAT con las exportaciones necesarias de DLL_FAKE: GetModuleHandle …**

- **Excepto en los módulos DLL_FAKE y DLL_REAL, se deben reemplazar todas las IATs que tengan exportaciones de DLL_REAL por las correspondientes de DLL_FAKE. La IAT de DLL_FAKE no se debe cambiar en caso de necesitar usar las exportaciones de DLL_REAL.**

PEB HOOKING

new w32 hooking skills

rooted 2010

- La DLL_FAKE debe exportar los mismos símbolos que la DLL_REAL, se necesita una DLL por cada versión, es decir, para usarlo con kernel32.dll necesitamos una DLL por cada versión de la misma que implemente nuevos símbolos exportados.
  - A veces es necesario pasar el control directamente a la DLL_REAL y no alterar el flujo original del proceso.

INDICE

new w32 hooking skills

rooted 2010

DWTF ENGINE

new w32 hooking skills

rooted 2010

- En el pasado para hacer PEB Hooking (o reemplazar una dll en disco) era necesario crear una plantilla .c y otra .h desde la dll real y luego compilarla.

- Esta herramienta exporta todos los simbolos de la dll real e importa todos sus exportaciones. Después crea un area con un JMP [ADDRESS] por cada export. Todo a nivel "binario".

- Puedes añadir o eliminar payloads con un simple IAT HOOKING en la DLL fake.

¡¡ DWTF + PEB HOOKING DEMO !!

INDICE

new w32 hooking skills

rooted 2010

E8 METHOD

new w32 hooking skills

rooted 2010

- **E8 METHOD simplemente permite obtener un ID de la API: E8 (CALL Opcode), no solo es sustituir un hook de tipo JMP por un hook de tipo CALL, es el concepto y/o la forma de implementar que un handler global ("One hook handler") pueda obtener el "hook_caller ID" en tiempo de ejecución". Puede haber muchos escenarios y formas de plasmar E8 Method, se seguira el siguiente contexto:**
  - **Escenario: Dado un hook de tipo Detour covencional se modificará para poder obtener el "hook_caller ID".**
  - **Forma: Se cambiará el salto al hook handler de tipo JMP por un salto de tipo CALL, y se reparará la pila para que todo funcione como en el método Detours convencional.**

E8 METHOD

new w32 hooking skills

rooted 2010

- **E8 METHOD simplemente permite obtener un ID de la API: E8 (CALL Opcode), no solo es sustituir un hook de tipo JMP por un hook de tipo CALL, es el concepto y/o la forma de implementar que un handler global ("One hook handler") pueda obtener el "hook_caller ID" en tiempo de ejecución". Puede haber muchos escenarios y formas de plasmar E8 Method, se seguira el siguiente contexto:**
  - **Escenario: Dado un hook de tipo Detour covencional se modificará para poder obtener el "hook_caller ID".**
  - **Forma: Se cambiará el salto al hook handler de tipo JMP por un salto de tipo CALL, y se reparará la pila para que todo funcione como en el método Detours convencional.**

E8 METHOD

- **Ejemplo de flujo Detours. Se ha introducido un hook de tipo Detour en la API Sleep de kernel32.dll**

1.El programa original (POC.exe) llama a Sleep

2.La primera instrucción de Sleep es una salto hacia el hook handler para dicha API.

3.El hook handler realiza las acciones para las que ha sido programado, llamando en un determinado momento a la API Sleep original usando el tramplín. Finalmente retorna al programa original como lo haría la API Sleep.

E8 METHOD

- **Ejemplo de flujo E8 Method llamando al trampolín. Se ha introducido un hook de tipo Detour en la API Sleep de kernel32.dll:**

1.El programa original (POC.exe) llama a Sleep

2.La primera instrucción de Sleep es un CALL hacia el hook handler para todos los hooks.

3.El hook handler realiza las acciones para las que ha sido programado, llamando en un determinado momento a la API Sleep original usando el trampolín. Finalmente retorna al programa original como lo haría la API Sleep en este caso.

E8 METHOD

- **Solo existen tres diferencias entre el método Detours convencional y E8 Method:**

1.En vez de un salto normal al hook handler se usa uno de tipo CALL.

2.En vez de un hook handler para cada hook se usará el mismo hook handler para todos los hooks.

3. Se debe extraer el hook caller ID introducido por el CALL que llama al hook handler de la pila para que no moleste y pueda ser todo como en el método Detours convencional.

INDICE

new w32 hooking skills

rooted 2010

ONE SAFE HOOK HANDLER

- **He definido Safe Hook Handler como un hook con dos características, debe ser:**

1.Invisible para el código legítimo: no alterar flags ni registros, no ser localizable usando estructuras internas del SO…

2.Evitar problemas de seguridad: sin stack buffers overflows al llamar desde el hook handler a un area hookeada…

ONE SAFE HOOK HANDLER

new w32 hooking skills

rooted 2010

- **Para conseguir o intentar ser invisible para el código legítimo, es necesario tener en cuenta:**

1.Evidencias de que existe un hook basadas en la arquitectura del microprocesador: registros, flags ...

2. Evidencias de que existe un hook no basadas en la arquitectura del microprocesador. Por ejemplo, un hook handler en Windows implementado en una DLL, para evitar una búsqueda usando el PEB, se podría desenlazar la DLL de la lista LDR_MODULE.

ONE SAFE HOOK HANDLER

- **Evidencias basadas en el microprocesador**

▪Problema: Comportamiento del API legítima usando como comprobación registros o flags del microprocesador que no modifique.

▪Solución: Cuando es llamado el hook handler se deben guardar todos los flags y registros que serán restaurados cuando se devuelva el control al código original.

ONE SAFE HOOK HANDLER

- **Evidencias basadas en el microprocesador**

▪Problema: Comportamiento del API legítima usando como comprobación registros o flags del microprocesador que modifique.

▪Solución: El hook handler debe llamar a la API original con los registros y flags iguales a los del código original, después de la llamada se debe guardar todos los flags y registros que serán restaurados cuando se devuelva el control al código original. En x86 sería suficiente con un PUSHAD y un PUSHFD cuando se llame al hook handler, cuando se llama a la API un POPFD y un POPAD y después de la llamada otro PUSHAD y PUSHFD, por último, cuando se devuelva el control al código original de nuevo hacer otro POPFD y POPAD de los valores que devolvió la llamada a la API.

ONE SAFE HOOK HANDLER

Para llevar acabo el E8 Method con un Safe Hook Handler, serán necesarios los siguientes elementos:

1.Biblioteca Detour adaptada a E8 Method para crear un solo hook handler para todos los hooks.

2.Una capa en ensamblador con dos propósitos:

1. **Permitir programar el hook handler en un lenguaje que no sea ensamblador como C/C++.**
2. **Evitar métodos de detección de hooks o anomalías, como el MOV EDI, ESP del visual studio para detectar corrupciones de pila**

3.Método para ocultar evidencias de que existe un hook handler que no tengan que ver con la arquitectura del microprocesado

4.Método para evitar stack buffers overflows y otros errores cuando se llama desde el hook handler a una API ya hookeada.

ONE SAFE HOOK HANDLER

new w32 hooking skills

rooted 2010

¡¡ MICROSOFT DETOURS DEMO !!

¡¡ EASY HOOK DEMO !!

¿TODO JUNTO?

new w32 hooking
skills
rooted 2010

- **PEB HOOKING + E8 METHOD + SAFE HOOK HANDLER**

AGRADECIMIENTOS

➢**pluf, griyo, tomac, slay, alon, blackngel, 48bits, bquintero, shearer, evilfingers, rooted…**

¿Preguntas? ¿Comentarios?

¡¡ Muchas gracias !!

kd> dt rooted!_ new_w32_hooking_skills_authors

+oxooo David Reguera García: _ROOTED_SPEAKER