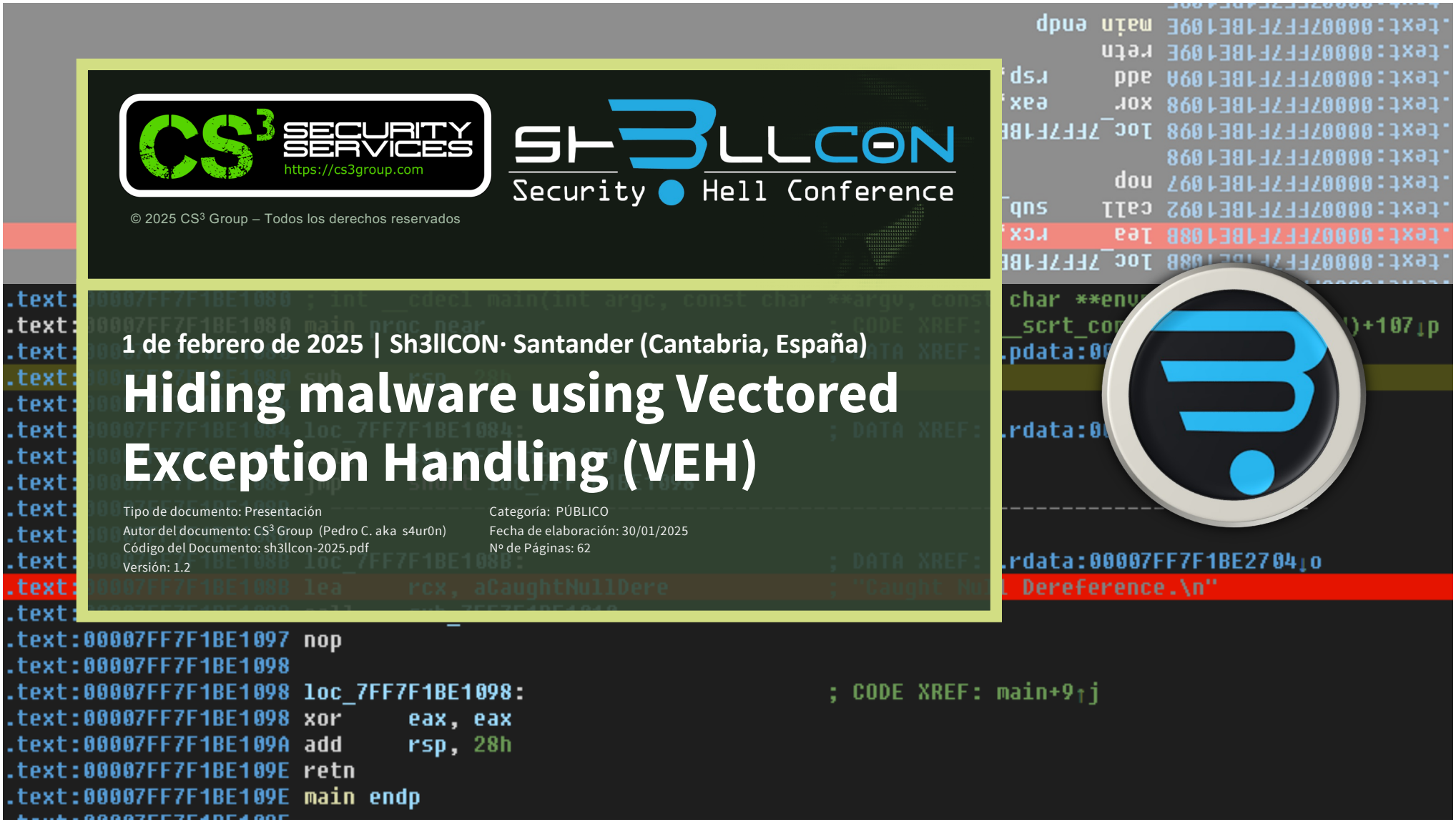


1 de febrero de 2025 | Sh3llCON· Santander (Cantabria, España)

Hiding malware using Vectored Exception Handling (VEH)

Tipo de documento: Presentación
Autor del documento: CS³ Group (Pedro C. aka s4ur0n)
Código del Documento: sh3llcon-2025.pdf
Versión: 1.2

Categoría: PÚBLICO
Fecha de elaboración: 30/01/2025
Nº de Páginas: 62

Whoami

```
class PedroC:
    def __init__(self):
        self.name =      'Pedro Candel'
        self.email =     's4ur0n@s4ur0n.com'
        self.web =       'https://www.s4ur0n.com'
        self.nick =      '@NN2ed_s4ur0n'
        self.company =   'CS³ Group'
        self.webcorp =   'https://cs3group.com'
        self.role =      'Security Researcher'
        self.work =      [ 'Reversing', 'Malware', 'Offensive Security', '…' ]
        self.groups =    [ 'mlw.re', 'OWASP', 'NetXploit', '...' ]
```

mlw.re

OWASP Open Web Application Security Project

CS³ SECURITY SERVICES https://cs3group.com

SHELLCON Security Hell Conference

CS³ Group

Formación en Seguridad
Cursos presenciales a medida impartidos en las instalaciones del cliente o las concertadas con prácticas reales desde el primer momento

Hacking Ético
Auditorías de caja negra, gris o blanca para aplicaciones web, sistemas y redes de comunicaciones

Análisis de Malware
Análisis de Malware automatizados y manuales con completos informes de comportamiento e indicadores de compromiso (IOC)

Ingeniería Inversa
Ingeniería Inversa para binarios de sistemas Windows de 32/64 bits, GNU/Linux de 32/64 bits, OSX Mach-O de 64 bits, ARM y firmwares

Exploiting
Desarrollo y adaptación de exploits para sistemas Windows de 32/64 bits, GNU/Linux de 32/64 bits, OSX Mach-O de 64 bits y Android

Desarrollo Seguro
Auditoría SAST, DAST, IAST y RASP para análisis de vulnerabilidades en el código de proyectos en Java, .Net, PHP, C/C++ y Cobol

Hardware Hacking
Análisis de vulnerabilidades en dispositivos hardware, sistemas embebidos y firmware con técnicas de ingeniería inversa

Seguridad en dispositivos móviles
Análisis estático, dinámico e instrumentación dinámica de aplicaciones Android (APK), iOS (IPA) y Windows Mobile (APPX)

Respuesta ante incidentes
Investigación remota de incidentes de seguridad, análisis de las situaciones y respuesta inmediata ante las amenazas

Forense
Adquisición y elaboración de informes periciales con garantía de imparcialidad y objetividad para todo tipo de sistemas de información

DevSecOps
Desarrollo, Seguridad y Operaciones en CSI (Continuous Security Integration) con pruebas automatizadas de seguridad para CI/CD

Intelligence
Recopilación, análisis y explotación de datos a gran escala con fuentes OSINT, SIGINT, HUMINT, Deep Web, redes P2P, etc.

SIGINT
Inteligencia de comunicaciones, análisis y auditoría de seguridad en señales y protocolos de radiofrecuencia (RF)

T.S.C.M.
Technical Surveillance Counter-Measures: Contramedidas electrónicas para detección y localización de dispositivos de escucha

Telecom
Análisis y auditoría GSM/3G/4G, implementación de servicios de operadores móviles virtuales (HLR, VLR, GGSN, Roaming voz y datos)

ATM
Análisis de vulnerabilidades, auditoría, forense, skimming, shimming y pruebas de blackbox para NCR, Hyosung, WRG, Diebold Nixdorf e Hitachi

PoS/TPV
Auditoría y cumplimiento de controles en terminales Verifone e Ingenico. Monitorización y transaccionabilidad completa según ISO 8583

LOPD/GPDR/Cumplimiento
LOPD, adaptación GPDR, ISO 27000, SGSI, análisis y gestión de riesgos, Políticas de seguridad, continuidad de negocio, ITIL, PCI DSS

CS³ SECURITY SERVICES https://cs3group.com

SH3LLCON Security Hell Conference

1. Control de excepciones estructurado

https://learn.microsoft.com/es-es/windows/win32/debug/structured-exception-handling

Control de excepciones estructurado

Una **excepción** es un evento que se produce durante la ejecución de un programa y requiere la ***ejecución de código fuera del flujo normal de control***.

Hay **dos tipos** de excepciones:

- **Excepciones de hardware**. La CPU inicia excepciones de hardware. Pueden resultar de la ejecución de determinadas secuencias de instrucciones, como la división por cero o un intento de acceder a una dirección de memoria no válida.

Control de excepciones estructurado

- **Excepciones de software**. Las aplicaciones o el sistema operativo inician explícitamente excepciones de software.

  Por ejemplo, el sistema puede detectar cuándo se especifica un valor de parámetro no válido.

Control de excepciones estructurado

El **control estructurado de excepciones** es un mecanismo para controlar las excepciones de hardware y software. Por lo tanto, el código ***controlará*** las ***excepciones de hardware y software de forma idéntica***.

Nos permite tener un **control completo** sobre el ***control de excepciones***, proporciona compatibilidad con los depuradores y se puede usar en todos los lenguajes de programación y máquinas.

Control de excepciones estructurado

El **control de excepciones vectoriales** es una ***extensión*** para el ***control estructurado*** de excepciones.

El sistema también admite el **control de terminación**, lo que le permite asegurarse de que cada vez que se ejecuta un cuerpo protegido del código, también se ejecuta un bloque especificado de código de terminación. Se ejecuta ***independientemente de cómo el flujo de control deja el cuerpo protegido***.

Control de excepciones estructurado

El **control estructurado de excepciones** está disponible principalmente a través de la ***compatibilidad con el compilador***.

El compilador de optimización de Microsoft C/C++ admite la palabra clave **`__try`** que identifica un ***cuerpo protegido*** del código, la palabra clave **`__except`** que identifica un ***controlador de excepciones*** y la palabra clave **`__finally`** que identifica un ***controlador de terminación***.

Control de excepciones estructurado

Aunque esta introducción usa ejemplos específicos del compilador de Microsoft C/C++, otros compiladores también proporcionan esta compatibilidad.

Es lo que se denomina **control de excepciones basado en marcos (Frame-based)**, que consta de:

- Cuerpo protegido del código.
- Expresión de filtro.
- Bloque de controlador de expresiones.

Control de excepciones estructurado

```
__declspec( noinline ) void ThrowNullPointerDereferenceException( void )
{
    volatile int* ptr = 0x0;
    *ptr = 0x1337;
}

int main( int argc, char** argv )
{
    __try
    {
        ThrowNullPointerDereferenceException();
    }
    __except ( EXCEPTION_EXECUTE_HANDLER )
    {
        printf( "Caught Null Dereference.\n" );
    }

    return 0;
}
```

Control de excepciones estructurado

Las excepciones se pueden **iniciar** por ***hardware*** o ***software***. Pueden producirse en **modo kernel** o en el código en **modo de usuario**. El control de excepciones estructurado es el único mecanismo para el **control de excepciones** en ***ambos modos***.

La ejecución de determinadas secuencias de instrucciones puede dar lugar a excepciones iniciadas por el Hardware. P.e. el hardware genera una infracción de acceso cuando un proceso intenta leer o escribir en una dirección virtual a la que no tiene el acceso adecuado.

Control de excepciones estructurado

Durante la ejecución de una rutina de Software (P.e. cuando se especifica un valor de parámetro no válido). Cuando esto sucede, un subproceso puede iniciar explícitamente una excepción mediante una llamada a la función **`RaiseException`**. Esta función permite al subproceso que llama especificar información que describe la excepción.

Una excepción puede ser **continuable** o **no continuable**. Una excepción no continuable ***no finaliza la aplicación***.

Control de excepciones estructurado

Cuando se produce una excepción de hardware o software, el procesador **detiene la ejecución** en el momento en el que se produjo la excepción ***y transfiere el control al sistema***.

En primer lugar, **el sistema *guarda el estado de la máquina del subproceso actual*** e ***información que describe la excepción***.

A continuación, **el sistema** intenta ***buscar un controlador de excepciones*** para controlar la excepción.

Control de excepciones estructurado

El **estado** se guarda en una **estructura `CONTEXT`** (***registro de contexto***) y le permite al sistema continuar la ejecución en el momento de la excepción si la excepción se controla correctamente.

La **descripción** de la excepción (***registro de excepción***) se guarda en una estructura de **`EXCEPTION_RECORD`**.

La información de los registros de contexto y de excepción está disponible con la función **`GetExceptionInformation`**.

Control de excepciones estructurado

Estructuras:

- **`CONTEXT`:** https://learn.microsoft.com/es-es/windows/win32/api/winnt/ns-winnt-arm64_nt_context
- **`EXCEPTION_RECORD`**: https://learn.microsoft.com/es-es/windows/win32/api/winnt/ns-winnt-exception_record

Control de excepciones estructurado

Modo usuario y orden de búsqueda:

- Si se depura el proceso, el sistema **notifica al depurador**.
- Si no se depura o el depurador no controla la excepción, el sistema intenta localizar el controlador frame-based en el marco de pila del subproceso y los anteriores.
- Si no se encuentra controlador frame-based o nadie controla la excepción, el sistema **notifica** al depurador ***una segunda vez***.
- Si no se depura o no se controla, el sistema proporciona el control (normalmente con **`ExitProcess`**).

Control de excepciones estructurado

Modo kernel y orden de búsqueda:

- El sistema busca en los ***marcos de pila*** de la pila del kernel en un intento de localizar un controlador de excepciones.
- Si un controlador no se encuentra o ningún controlador controla la excepción, el sistema se apaga como si se hubiera llamado a la función **`ExitWindows`.**

Control de excepciones estructurado

Un **controlador de terminación** garantiza que se ejecute un bloque de código específico cada vez que el flujo de control sale de un cuerpo de código protegido determinado.

Consta de los siguientes elementos:

- Un ***cuerpo protegido*** del código
- ***Bloque de código de terminación*** que se va a ejecutar cuando el flujo de control deja el cuerpo protegido.

Control de excepciones estructurado

Los controladores de terminación se declaran en la sintaxis específica del lenguaje.

Con el compilador de optimización de Microsoft C/C++, se implementan mediante **`__try`** y **`__finally`**.

Siempre que se ejecute el cuerpo protegido, se ejecutará el bloque de código de terminación, ***excepto*** si finaliza con **`ExitThread`** o **`ExitProcess`** ***desde el cuerpo protegido***.

Control de excepciones estructurado

Los **controladores de excepciones vectoriales** son una ***extensión*** para el ***control de excepciones estructurado***.

Los controladores vectoriales ***no están basados en fotogramas***, por lo que puede agregar un controlador al que **se llamará independientemente de dónde se encuentre** en un marco de llamada.

Se llaman en el **orden en que se agregaron**, después de que el depurador reciba la ***primera notificación***, pero **antes** de que el sistema comience a ***desenredar la pila***.

Control de excepciones estructurado

Para agregar un **controlador de excepciones vectoriales**, se emplea la función **`AddVectoredExceptionHandler`**. Para quitarlo, se usa **`RemoveVectoredExceptionHandler`**.

Para agregar un **controlador de continuación vectorial**, se emplea la función **`AddVectoredContinueHandler`**. Para quitarlo, se usa **`RemoveVectoredContinueHandler`**.

Control de excepciones estructurado

```
int main() {
  AddVectoredExceptionHandler(1, MyVEHHandler);
  int except = 5;
  except = except / 0;
  return 0;
}
```

Control de excepciones estructurado

```
LONG NTAPI MyVEHHandler(PEXCEPTION_POINTERS ExceptionInfo) {
  printf("MyVEHHandler (0x%x)\n", ExceptionInfo->ExceptionRecord->ExceptionCode);

  if (ExceptionInfo->ExceptionRecord->ExceptionCode == EXCEPTION_INT_DIVIDE_BY_ZERO) {
    printf("  Divide by zero at 0x%p\n", ExceptionInfo->ExceptionRecord->ExceptionAddress);
    ExceptionInfo->ContextRecord->Eip += 2;
    return EXCEPTION_CONTINUE_EXECUTION;
  }

  return EXCEPTION_CONTINUE_SEARCH;
}
```

2. Controlador de Excepción Vectorial

VEH (Vectored Exception Handling)

Controlador de Excepción Vectorial

Cuando ocurre una **excepción de CPU**, el Kernel llamará a la función **`KiDispatchException`** (**ring0**).

El Kernel ***seguirá esta excepción*** mediante el método de **NTDLL `KiUserExceptionDispatcher`** (**ring3**).

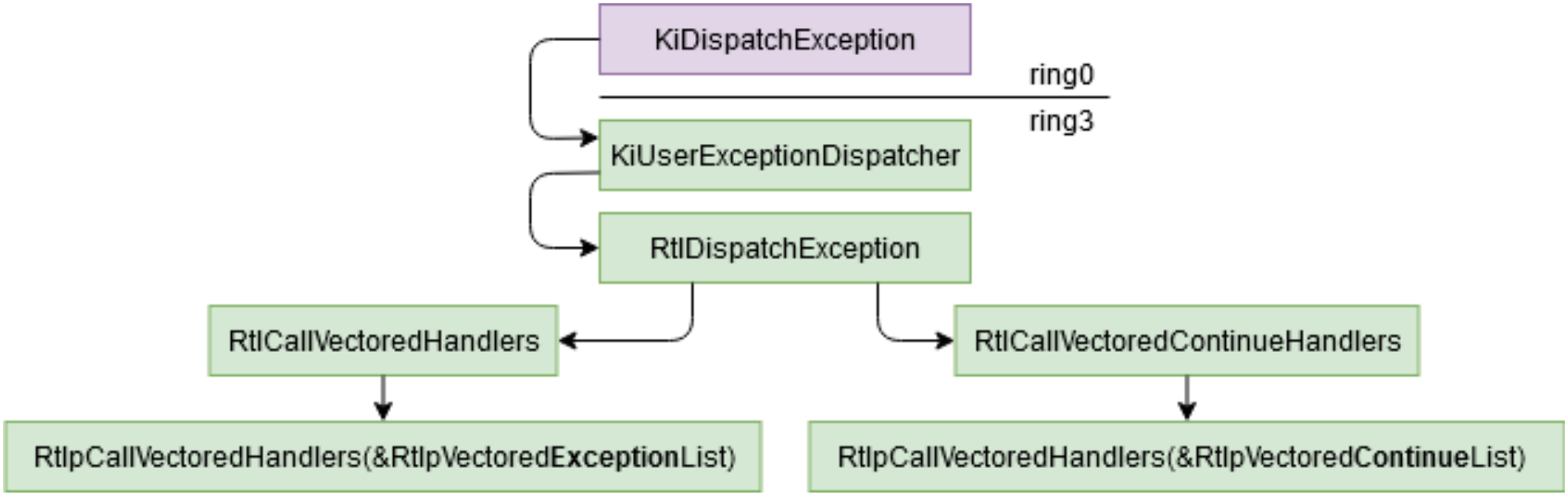
Esta función llamará a **`RtlDispatchException`** que ***intentará manejarla a través del VEH***.

Controlador de Excepción Vectorial

Para hacerlo, **lee** la ***lista encadenada del VEH*** a través de **`RtlCallVectoredHandlers`** y **llamará a cada manejador** hasta que uno devuelva **`EXCEPTION_CONTINUE_EXECUTION`**.

Si un manejador devuelve **`EXCEPTION_CONTINUE_EXECUTION`**, se llama a la función **`RtlCallVectoredContinueHandlers`** y **llamará a todos** los ***manejadores de excepciones de continuación***.

Controlador de Excepción Vectorial



Controlador de Excepción Vectorial

Los controladores VEH son importantes porque los **controladores SEH** se llaman únicamente ***si ningún controlador VEH ha capturado la excepción***, por lo que podría ser el mejor método para capturar todas las excepciones si no se quiere hookear **`KiUserExceptionDispatcher`**.

Controlador de Excepción Vectorial

La lista VEH es una **lista enlazada circular** con los **punteros de funciones de controlador *codificados***.

Los controladores de excepción están **codificados** con una **cookie de proceso**.

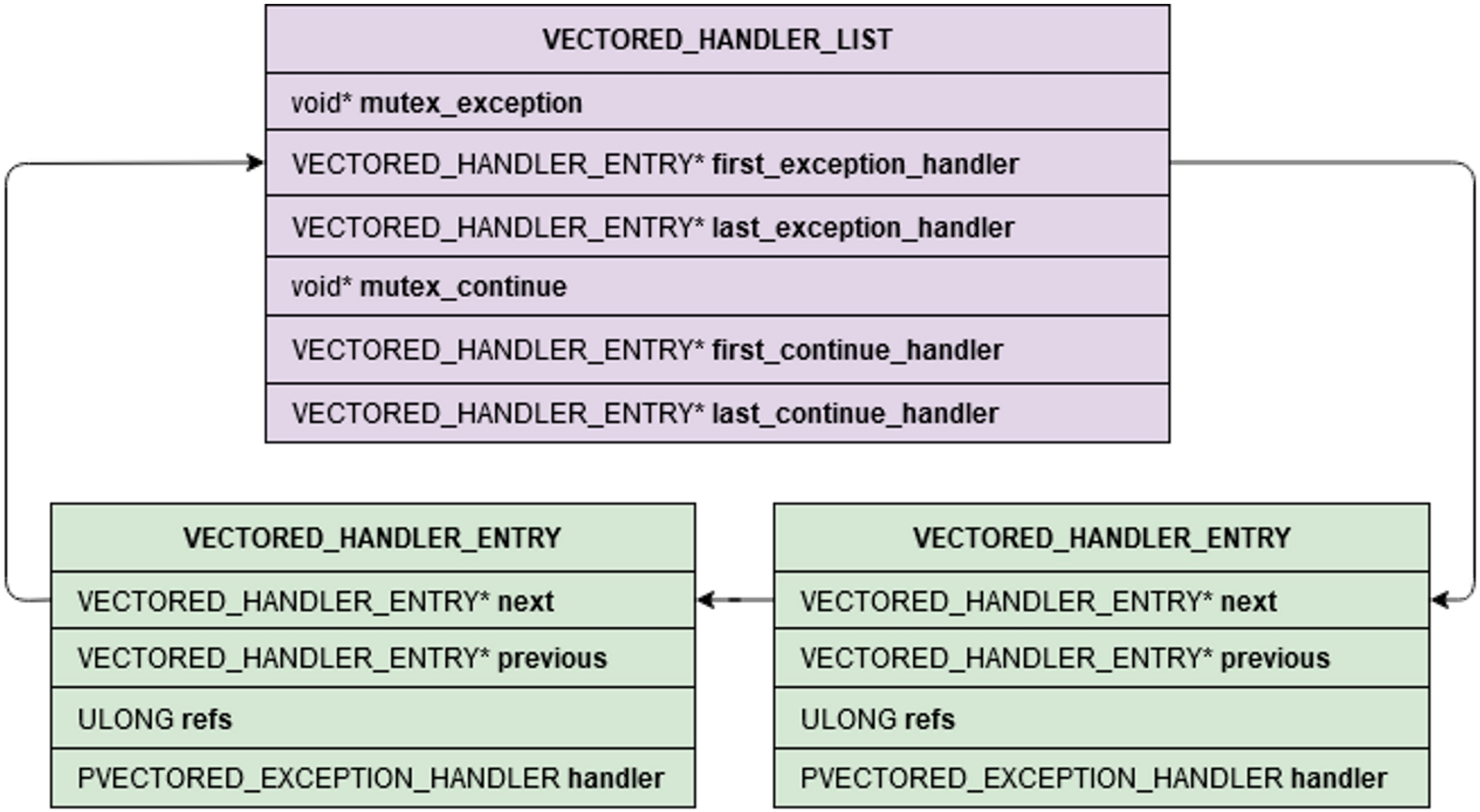
Pueden decodificarse fácilmente.

Controlador de Excepción Vectorial

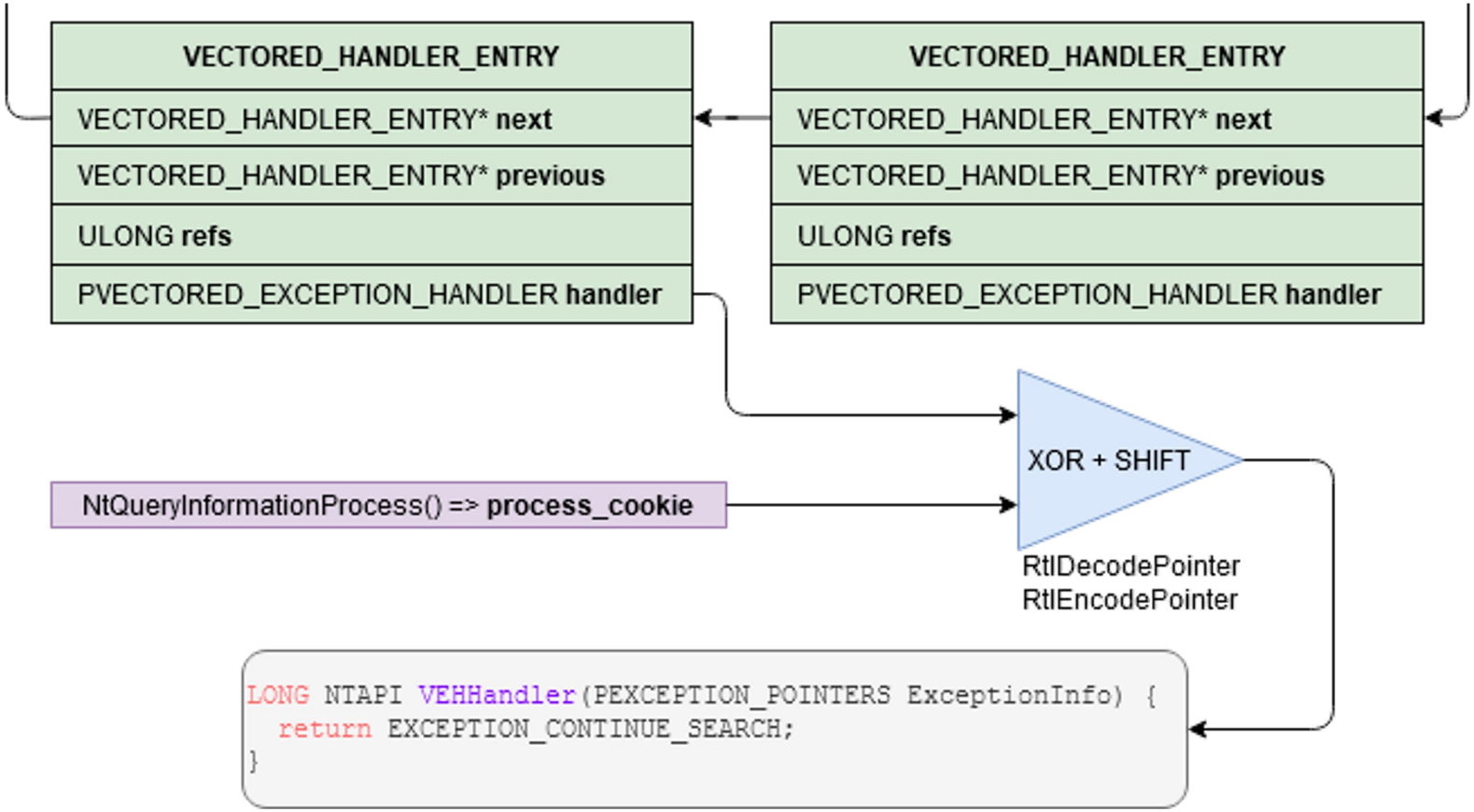
Si queremos dumpear el VEH **dentro de nuestro propio proceso**, se puede usar **`DecodePointer`** y no hay que preocuparse por la cookie.

Si es un poceso remoto, puede emplearse **`DecodeRemotePointer`**, que hay que crear un puntero de función con **`GetModuleHandle("kernel32.dll")`** y **`GetProcAddress("DecodeRemotePointer")`**.

Controlador de Excepción Vectorial



Controlador de Excepción Vectorial



Controlador de Excepción Vectorial

Encontrar el Offset de la lista VEH

Incluso pudiendo encontrar el símbolo **`LdrpVectorHandlerList`** en el PDB de NTDLL, **no existe API oficial para obtenerlo** fácilmente.

Los EDRs suelen intentar hookear con un puntero a **`RtlpAddVectoredHandler`**

3. Voidgate

Bypass AV/EDR memory scanners with VEH (Vectored Exception Handling)

Voidgate

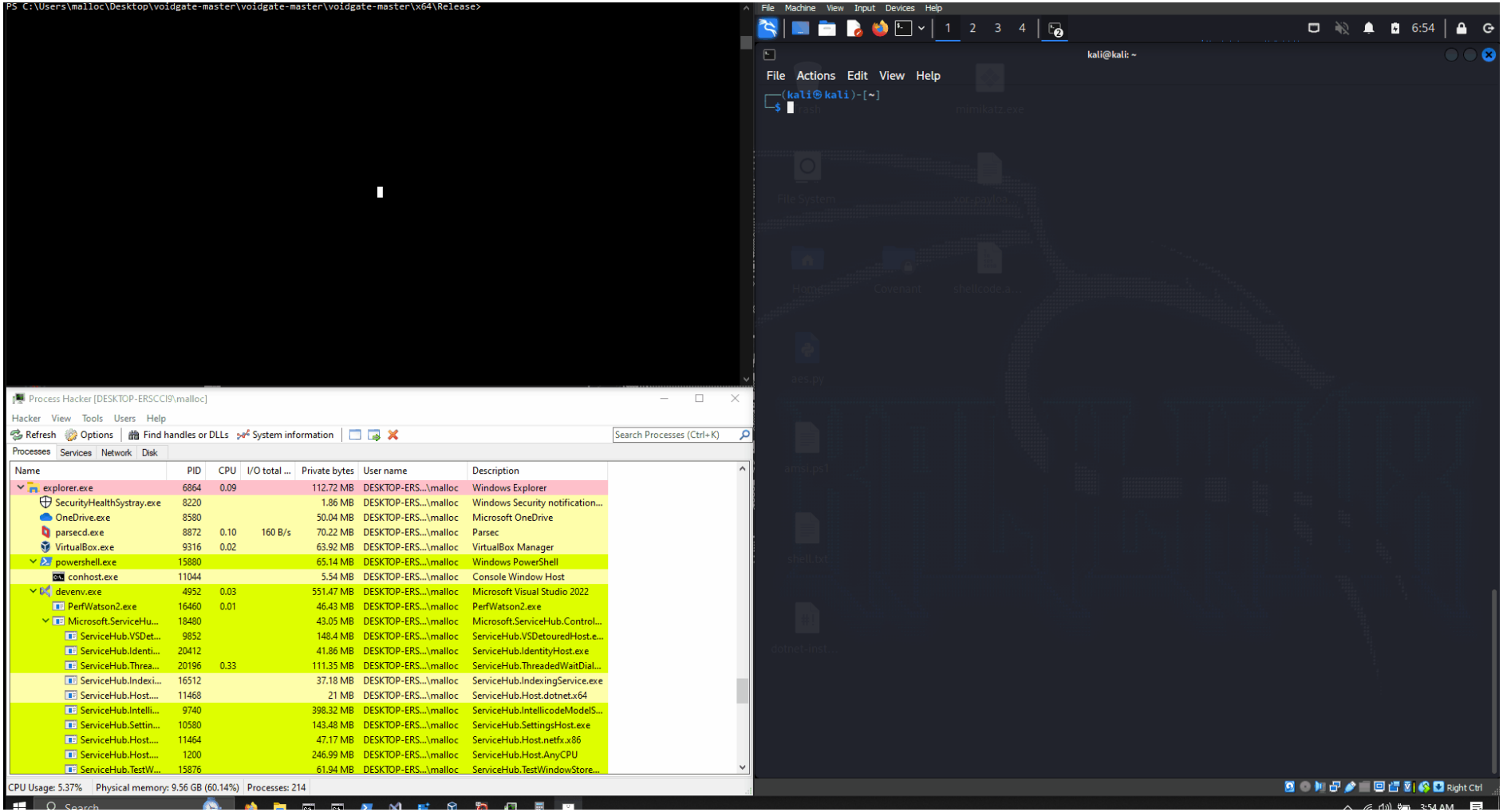
No importa el **algoritmo** o la **resistencia** con la que se ***cifre una shellcode***... Se necesita descifrarla completamente para ejecutarla.

Los **AV/EDR** se aprovechan de ello para **detectar los payloads** mediante ***comparación de patrones***.

¿Y si sólo se descifra **una instrucción a la vez**, se **ejecuta** y se **vuelve a cifrar antes** de pasar a la ***siguiente instrucción***? Esa técnica es la conocida como **VoidGate**.

Voidgate

https://github.com/vxCrypt0r/Voidgate



3. Voids4ugate

Redefining Voidgate

Voids4ugate

Hardware Breakpoints

Cuando se utiliza un debugger para depurar un programa, establecemos **Breakpoints** que pausan la ejecución en una dirección. El debugger debe comprobar el **RIP/EIP/IP** para ver si coincide con alguna entrada de la lista.

La CPU puede establecer **Hardware Breakpoints**. Estos no necesitan software específico en el área de usuario y son independientes del debugger.

Voids4ugate

Cuando **se alcanza el HWBP**, la CPU:

- **Detiene la ejecución** del subproceso.
- **Genera una excepción**.
- Permite que el desarrollador **gestione la excepción**.

Todo este proceso se logra a través de los "**registros de depuración**".

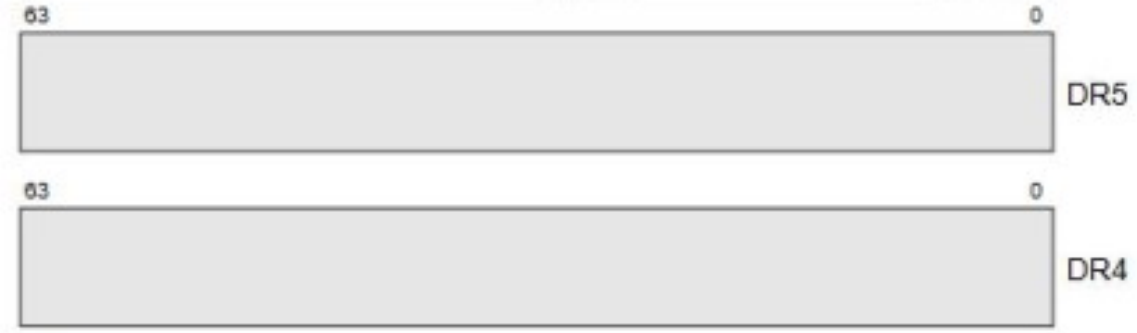
Voids4ugate

Registros de depuración (Max. 4 HWBP)

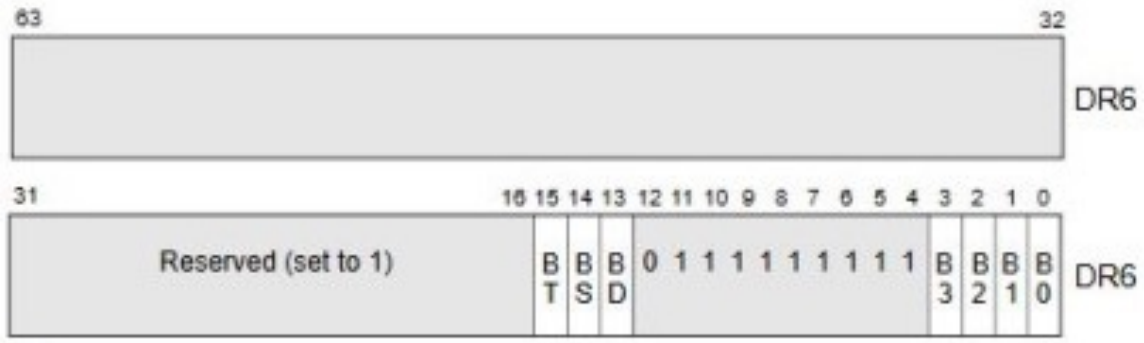
Figure 17-2. DR6/DR7 Layout on Processors Supporting Intel® 64 Architecture

Voids4ugate

Reservados si está activo el flag DE en CR4, sino, **asignados a DR6 y DR7** (condiciones normales)



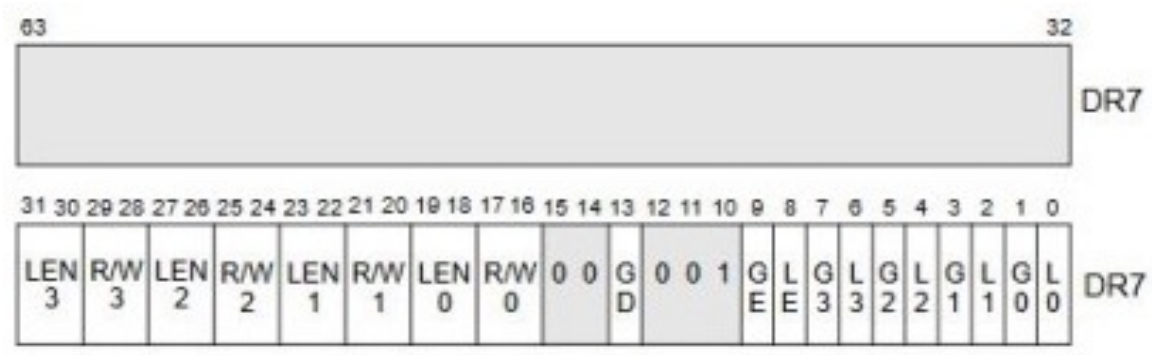
Registro de control de depuración (información sobre las condiciones en que se activó el evento del HWBP). Sus campos deben **eliminados por el depurador** antes de retornar al subproceso interrumpido.



Condición **BP detectada**

BT: Task switch (el trap flag del TSS está fijado)
BS: Single Step (excepción generada por single-stepping Execution mode)
BD: Debug Register Access Detected (la siguiente instrucción accederá a uno de los registros DR0-DR7)

Registro de control de depuración (Habilita/Deshabilita BPs y setea las condiciones de los BPs).



L0-L3: Local BP Enable
G0-G3: Global BP Enable
GD: General Detect Enable
LE: Local Exact BP Enable
GE: Global Exact BP Enable
R/W0-R/W3: Read/Write BP
LEN0-LEN3: BP location size

Voids4ugate

Manejadores de excepciones vectorizados

- El manejador de excepciones vectorizado (VEH) es una extensión de SEH.
- Son globales.
- Se puede registrar un VEH para manejar selectivamente una excepción particular y el resto con SEH.
- Reciben punteros al contexto del subproceso y lo datos de excepción.

Voids4ugate

¿Qué significa esto?

Los VEH **pueden realizar cambios en el contexto de ejecución** de un hilo ***mientras manejan la excepción***.

Voids4ugate

Voidgate es una técnica que ***descifra dinámicamente*** solo unas pocas instrucciones de una shellcode a la vez (al menos 1, como máximo 3 ó 4), lo ***ejecuta*** y luego ***lo vuelve a cifrar antes de continuar*** y vuelve a hacer lo mismo con las siguientes instrucciones.

Con Voidgate, dado que solo se **descifran entre 0 y 4 instrucciones** de toda la shellcode en cualquier momento, el escaneo de la memoria fallará.

Voids4ugate

Es muy difícil escribir **firmas confiables para 1 a 4 instrucciones** de una shellcode porque causarán una cantidad muy alta de **falsos positivos**.

Y ese es precisamente el ***objetivo de Voidgate***.

Voids4ugate

Funcionamiento y adaptación realizada:

- La shellcode debe estar ***cifrada*** (requisito).
- Se **copia** a un área de ***memoria R/W*** (***no*** *e**X**ecutable*)
- Se crea nuevo ***hilo suspendido*** apuntando al comienzo de dicha región de memoria.
- Se **modifica** el contexto del hilo: **`DR0`** tiene dicha dirección y **`DR7`** lo habilita localmente.

Voids4ugate

- Se **registra un VEH** que maneja **`EXCEPTION_SINGLE_STEP`** generado desde la shellcode y pasa cualquier otra excepción a los manejadores siguientes.

  Como se activa antes que la ejecución real, aquí es donde vamos a descifrar la(s) instrucción(es) de destino en las que se tiene el Hardware Breakpoint.

Voids4ugate

- Se **establece** el ***flag TF (tramp)*** y de ***reanudación (RF)*** en el contexto del hilo dentro de **`EFlags`**.

  `TF` hace que la excepción se vuelva a disparar en la siguiente instrucción incluso sin registros configurados en ella (`DR0` sólo tenía la dirección de comienzo y la segunda es el desplazamiento +1).

  `RF` se configura para que la ejecución avance y no quede en bucle. Resumen: recorre toda la shellcode (una instrucción a la vez).

Voids4ugate

- El **VEH devuelve *EXCEPTION_CONTINUE_EXECUTION*** luego el hilo continua la ejecución de las instrucciones descifradas.

- Si la excepción se genera en otro lugar que no sea la primera instrucción, el ***VEH vuelve a cifrar*** las instrucciones anteriores antes de descifrar.

Voids4ugate

- El **VEH descifra/cifra N bytes** de la shellcode a la vez. N es la longitud máxima de una instrucción en ensamblador en la arquitectura de destino (16 bytes en sistemas x64). Si los N bytes contienen 3 instrucciones, procesa sólo 3 o incluso 1, 2…

- El **RIP** (que se puede ***derivar del contexto del hilo*** recibido por el VEH), indica la **dirección que debe ejecutar** el hilo ***tras manejar la excepción*** el VEH. P.e. si la excepción se activa desde `0xAB`, `RIP=0xAB` y descifrará hasta `RIP+16=0xBB` antes que VEH regrese.

Voids4ugate

- Si la excepción se activa desde `0xAC`, el **VEH vuelve a cifrar** desde `0xAB-0xBB` antes de descifrar los datos desde `0xAC-0xBC`. Por eso, para descifrar ***hay que guardar el RIP*** para poder utilizarlo en la siguiente etapa de re-cifrado del siguiente ciclo.

- Se **reanuda el hilo** suspendido y se **activa el VEH** en cada instrucción.

Voids4ugate

Problemas y limitaciones

- Hay un problema obvio con la implementación: si la shellcode ***hace referencia a cualquier otra parte de sí misma***, **fallaría**, ya que esta referencia devolvería datos cifrados, no los datos descifrados reales.

- Voidgate o sus derivados **sólo pueden usarse** para **shellcodes independiente de la posición (PIC)**, haciendo de ellas un excelente método de first-stage para operaciones de Red Team.

4. Bonus track: Voids4ugateSteroids

Adding steroids to Voidgate

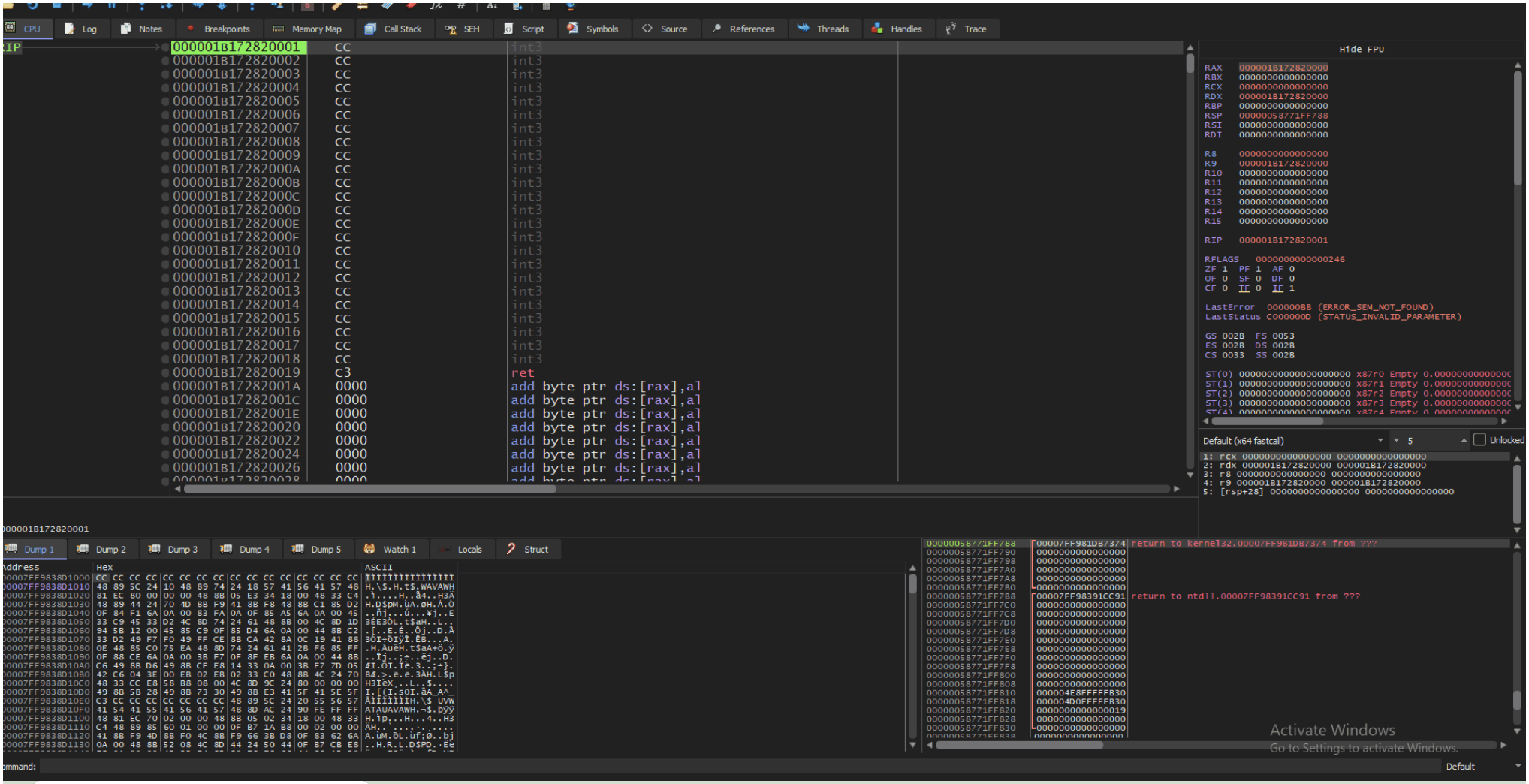
Voids4ugateSteroids

~~Problemas y limitaciones~~

- ~~Hay un problema obvio con la implementación: si la shellcode *hace referencia a cualquier otra parte de sí misma*, **fallaría**, ya que esta referencia devolvería datos cifrados, no los datos descifrados reales.~~

- Voidgate o sus derivados ~~**sólo pueden usarse** para **shellcodes independiente de la posición (PIC)**~~, haciendo de ellas un excelente método ~~de first-stage~~ para operaciones de Red Team.

Depurando la shellcode



¿Preguntas?

¡Muchas gracias!