Bypass Hardware Breakpoint Protection

by Mattwood FRET[1]

November 4, 2005

[1]mattwood9@gmail.com

Contents

Chapter 1

Hardware breakpoint

Hardware breakpoints or BPM(Breakpoint on Memory) use the Debug Registers. The way of concretisation it depending of the processor. In the architectural x86 (+386) 4 debug registers(DR0-DR3) exists in the processor. So we can only set 4 BPM. The DR4 and DR5 are reserved registers, the DR6 register is for the debugger and the DR7 register still using for the type of break for the DR0-DR3 registers.

1.1 Debug Address Registers (DRO-DR3)

Each of these registers contains the linear address associated with one of four breakpoint conditions. Each breakpoint condition is further defined by bits in DR7.

The debug address registers are effective whether or not paging is enabled. The addresses in these registers are linear addresses. If paging is enabled, the linear addresses are translated into physical addresses by the processor's paging mechanism. If paging is not enabled, these linear addresses are the same as physical addresses.

Note that when paging is enabled, different tasks may have different linear-to-physical address mappings. When this is the case, an address in a debug address register may be relevant to one task but not to another. For this reason the 80386 has both global and local enable bits in DR7. These bits indicate whether a given debug address has a global (all tasks) or local (current task only) relevance.[1]

[1]http://www.online.ee/ andre/i80386/Chap12.html

1.2 Debug Control Register (DR7)

The debug control register shown in Figure 12-1 both helps to define the debug conditions and selectively enables and disables those conditions.

For each address in registers DR0-DR3, the corresponding fields R/W0 through R/W3 specify the type of action that should cause a breakpoint. The processor interprets these bits as follows:

00 – Break on instruction execution only
01 – Break on data writes only
10 – undefined
11 – Break on data reads or writes but not instruction fetches

Fields LEN0 through LEN3 specify the length of data item to be monitored. A length of 1, 2, or 4 bytes may be specified. The values of the length fields are interpreted as follows:

00 – one-byte length
01 – two-byte length
10 – undefined
11 – four-byte length

If RWn is 00 (instruction execution), then LENn should also be 00. Any other length is undefined.

The low-order eight bits of DR7 (L0 through L3 and G0 through G3) selectively enable the four address breakpoint conditions. There are two levels of enabling: the local (L0 through L3) and global (G0 through G3) levels. The local enable bits are automatically reset by the processor at every task switch to avoid unwanted breakpoint conditions in the new task. The global enable bits are not reset by a task switch; therefore, they can be used for conditions that are global to all tasks.

The LE and GE bits control the "exact data breakpoint match" feature of the processor. If either LE or GE is set, the processor slows execution so that data breakpoints are reported on the instruction that causes them. It

is recommended that one of these bits be set whenever data breakpoints are armed. The processor clears LE at a task switch but does not clear GE. [2]

[2]http://www.online.ee/ andre/i80386/Chap12.html

1.3 Debug Register Table

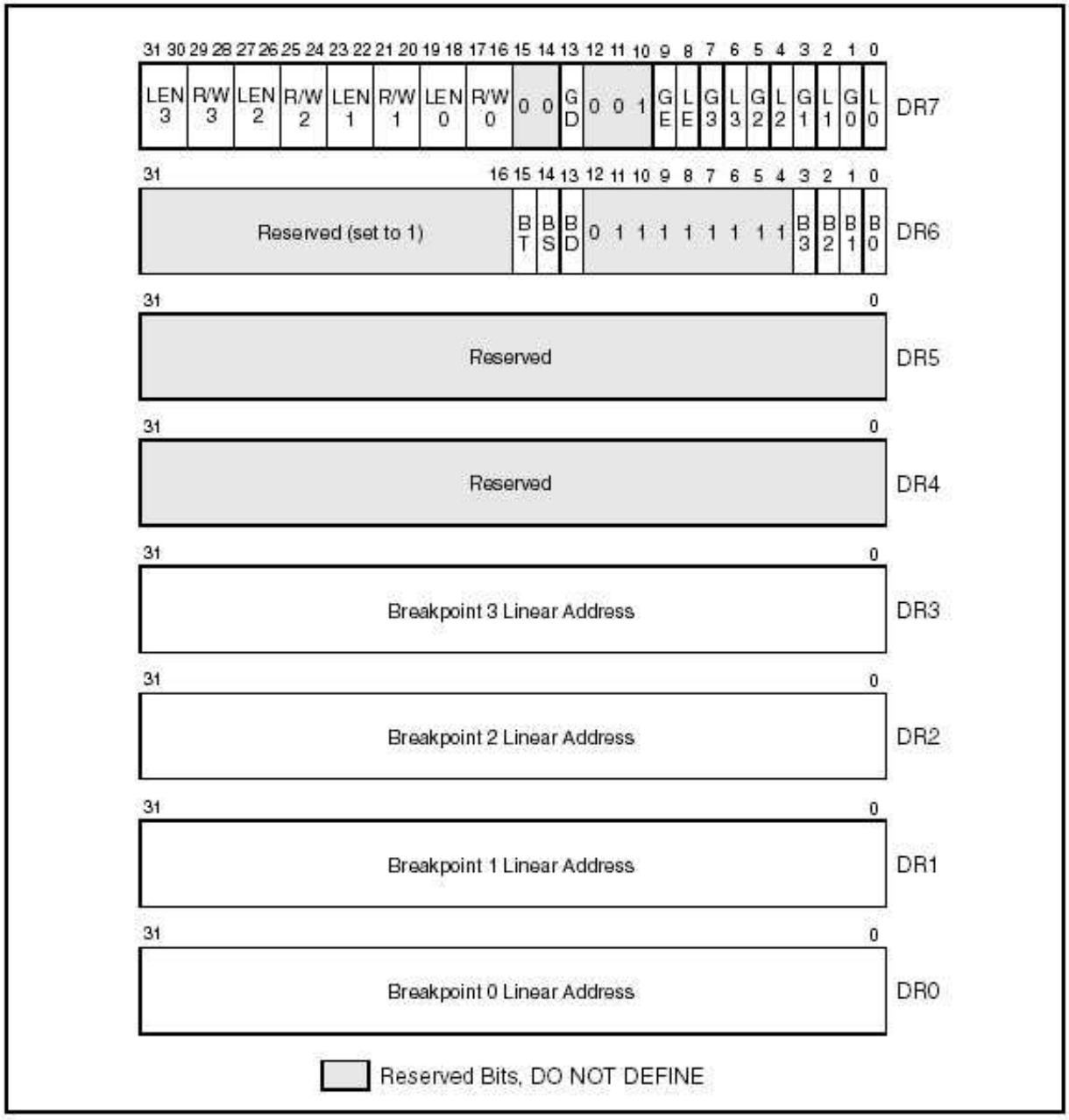
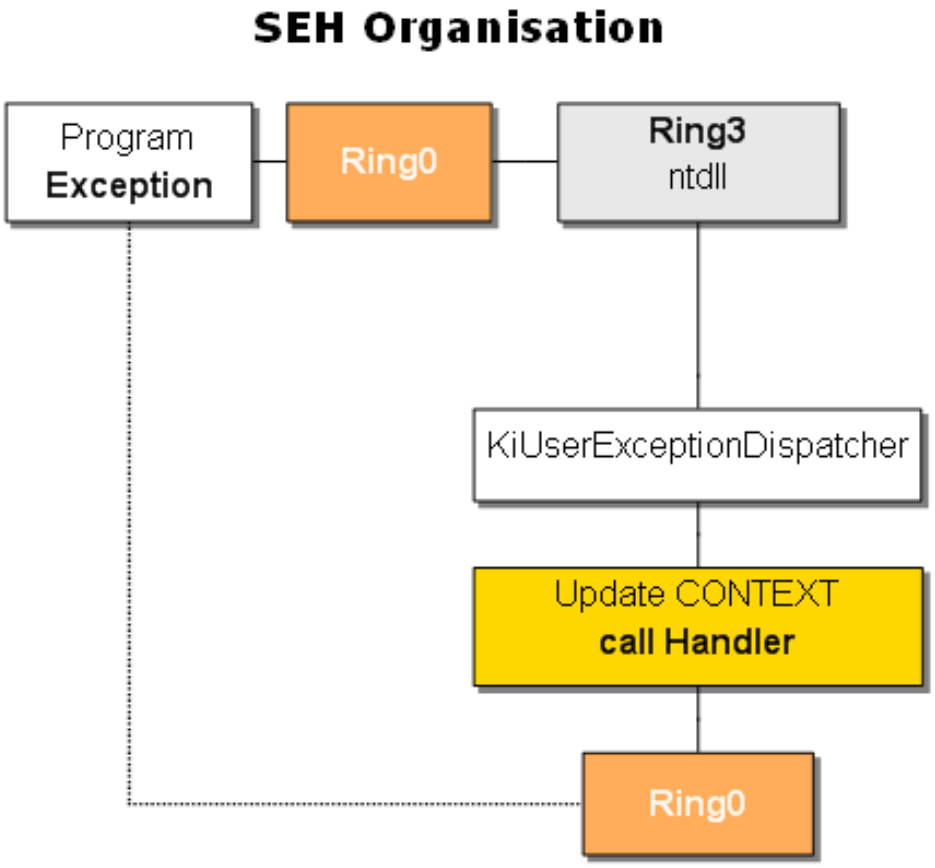


Figure 15-1. Debug Registers

Chapter 2

Anti Hardware Breakpoint

2.1 How the system manages the SEH?



sequently we have access to the DRx register.

2.2 Explanation

The DR0,DR1,DR2,DR3,DR6 registers are set to 0 and the DR7 to 155h.
But why to 155h?
Note: 155h = 0101010101b

DR7

G	L	G	L	G	L	G	L	G	L
E	E	3	3	2	2	1	1	0	0
0	1	0	1	0	1	0	1	0	1

L0 = 1 ; Active local DR0 (only for this process). We need to set this value else windows won't use the new DR0.
G0 = 0 ; Disactive global DR0 (for all tasks)

L1 = 1 ; Active local DR1 (only for this process). We need to set this value else windows won't use the new DR1.
G1 = 0 ; Disactive global DR1 (for all tasks)

L2 = 1 ; Active local DR2 (only for this process). Same.
G2 = 0 ; Disactive global DR2 (for all tasks)

L3 = 1 ; Active local DR3 (only for this process). Same.
G3 = 0 ; Disactive global DR3 (for all tasks)

LE = 1 ; Actived for a compatibility problem

2.3 Implemention

Using SEH to erase the current BPM is a very common technique, and is very easy to recognise.
A sample of Anti-Hardware Breakpoint:

```
push    offset seh_handler          ; New Handler
xor     eax, eax
push    dword ptr fs:[eax]          ;Old Handler
mov     dword ptr fs:[eax],esp      ;The TIB pointe on our structure
xor     dword ptr [eax],eax         ; Exception
pop     dword ptr fs:[eax]          ;Restore the TIB with the previous Handler
lea     esp, dword ptr [esp+4]      ;Ajust ESP
call    ExitProcess
retn
seh_handler proc ExceptionRecord:DWORD, EstablisherFrame:DWORD,\
        ContextRecord:DWORD, DispatcherContext:DWORD
mov     eax, ContextRecord
assume  eax:ptr CONTEXT
mov     dword ptr [eax].iDr0, 0     ; DR0 = 0
mov     dword ptr [eax].iDr1, 0     ; DR1 = 0
mov     dword ptr [eax].iDr2, 0     ; DR2 = 0
mov     dword ptr [eax].iDr3, 0     ; DR3 = 0
mov     dword ptr [eax].iDr6, 0     ; DR6 = 0
mov     dword ptr [eax].iDr7, 155h  ; DR7 = 155h
add     dword ptr [eax].regEip, 2   ; EIP += 2
ret
seh_handler endp
```

```
CONTEXT STRUCT
ContextFlags        DWORD                   ?
iDr0                DWORD                   ?
iDr1                DWORD                   ?
iDr2                DWORD                   ?
iDr3                DWORD                   ?
iDr6                DWORD                   ?
iDr7                DWORD                   ?
FloatSave           FLOATING_SAVE_AREA ¡¿
regGs               DWORD                   ?
regFs               DWORD                   ?
regEs               DWORD                   ?
regDs               DWORD                   ?
regEdi              DWORD                   ?
regEsi              DWORD                   ?
regEbx              DWORD                   ?
regEdx              DWORD                   ?
regEcx              DWORD                   ?
regEax              DWORD                   ?
regEbp              DWORD                   ?
regEip              DWORD                   ?
regCs               DWORD                   ?
regFlag             DWORD                   ?
regEsp              DWORD                   ?
regSs               DWORD                   ?
ExtendedRegisters   db                      MAX dup(?)
CONTEXT ENDS
```

Chapter 3

Anti AntiHardware Breakpoint

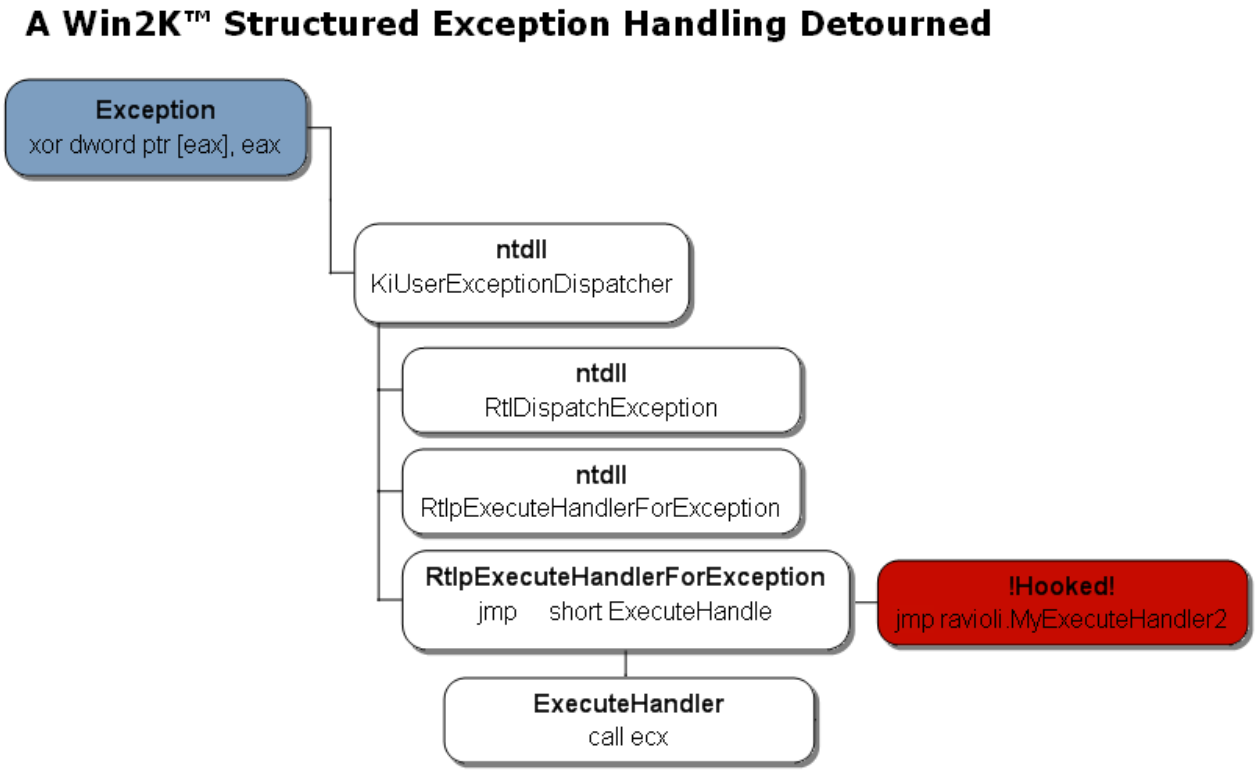
3.1 How to hook a SEH?

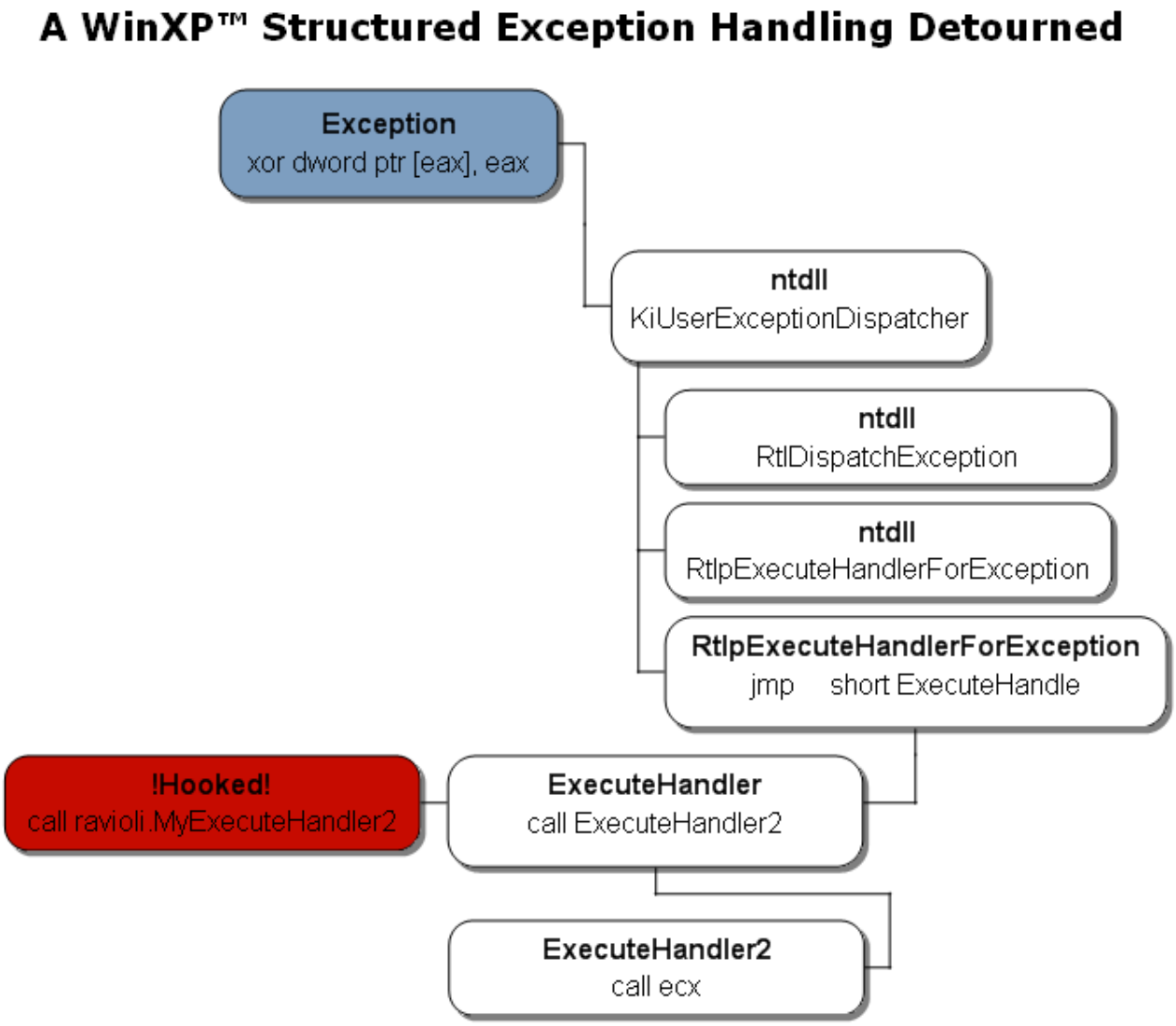
In the previous chapter we saw that the system passes the EXCEPTION to ntdll.KiUserExceptionDispatcher in Ring3. Therefore, all we need is to hook the function that calls our HANDLER.

Those functions differs in every OS.

In the next pages we will see two flow chart of hooked KiUserExceptionDispatcher.

In Win2K I replace the jmp ExecuteHandler to jmp MyExecuteHandler.
In WinXP I replace the call ExecuteHandler2 to call MyExecuteHandler2


A Win2K™ Structured Exception Handling Detourned
Exception
xor dword ptr [eax], eax
ntdll
KiUserExceptionDispatcher
ntdll
RtlDispatchException
ntdll
RtlpExecuteHandlerForException
RtlpExecuteHandlerForException
jmp short ExecuteHandle
!Hooked!
jmp ravioli.MyExecuteHandler2
ExecuteHandler
call ecx


A WinXP™ Structured Exception Handling Detourned
Exception
xor dword ptr [eax], eax
ntdll
KiUserExceptionDispatcher
ntdll
RtlDispatchException
ntdll
RtlpExecuteHandlerForException
RtlpExecuteHandlerForException
jmp short ExecuteHandle
!Hooked!
call ravioli.MyExecuteHandler2
ExecuteHandler
call ExecuteHandler2
ExecuteHandler2
call ecx

3.2 MyExecuteHandler

This piece of code is a sample of a modified NewExecute2 function.

```
int _stdcall NewExecute2(DWORD ExceptionRecord, DWORD
EstablisherFrame, LPCONTEXT ContextRecord, DWORD Dis-
patcherContext,SEHandler Handler) {

DRx.Dr0 = ContextRecord-¿Dr0;
DRx.Dr1 = ContextRecord-¿Dr1;
DRx.Dr2 = ContextRecord-¿Dr2;
DRx.Dr3 = ContextRecord-¿Dr3;
DRx.Dr6 = ContextRecord-¿Dr6;
DRx.Dr7 = ContextRecord-¿Dr7;

printf("Eip: 0x%08X // SEH Handler : 0x%08X\n",ContextRecord-
>Eip, Handler);

Handler(ExceptionRecord, EstablisherFrame, ContextRecord, Dis-
patcherContext);

ContextRecord->Dr0 = DRx.Dr0;
ContextRecord->Dr1 = DRx.Dr1;
ContextRecord->Dr2 = DRx.Dr2;
ContextRecord->Dr3 = DRx.Dr3;
ContextRecord->Dr6 = DRx.Dr6;
ContextRecord->Dr7 = DRx.Dr7;

return 1;

}
```

NewExecute(2) is a very basic function. Therefore, it needs no effort to rewrite. The arguments are similar to the Handler's.

```
typedef int (__cdecl *SEHandler)(DWORD ExceptionRecord, DWORD
EstablisherFrame, LPCONTEXT ContextRecord, DWORD Dis-
patcherContext);
```

3.3 Standalone Concept

```
/*************************************************************************
Mattwood^FRET 2oo5 // Magic Source haha //
Standalone Concept // mattwood9@gmail.com
*************************************************************************/

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct _DebugRegister {
DWORD Dr0;
DWORD Dr1;
DWORD Dr2;
DWORD Dr3;
DWORD Dr6;
DWORD Dr7;
} DebugRegister;

DebugRegister DRx;

typedef int (__cdecl *SEHandler)(DWORD ExceptionRecord, DWORD
EstablisherFrame, LPCONTEXT ContextRecord, DWORD Dis-
patcherContext);
int _stdcall NewExecute2(DWORD ExceptionRecord, DWORD
EstablisherFrame, LPCONTEXT ContextRecord, DWORD Dis-
patcherContext,SEHandler Handler);

int main(int argc, char **argv) {
unsigned int i, j;

DWORD pKiUserExceptionDispatcher, pRtlDispatchException,
pRtlpExecuteHandlerForException, pExecuteHandler;
DWORD pNewExecute2;
DWORD lpWrByte;

pKiUserExceptionDispatcher = (DWORD )GetProcAddress(LoadLibrary("ntdll.dll"),
"KiUserExceptionDispatcher");
printf("KiUserExceptionDispatcher : %08X\n\n", pKiUserEx-
ceptionDispatcher);
```

```
for(i=0; ((BYTE *)(pKiUserExceptionDispatcher))[i] != 0xE8;
i++)
continue;

printf("call RtlDispatchException : %08X\n", pKiUserExcep-
tionDispatcher+i);
pRtlDispatchException = pKiUserExceptionDispatcher+i;

pRtlDispatchException += ( (DWORD *)(pRtlDispatchExcep-
tion+1) )[0] + 5;

printf("RtlDispatchException: %08X
n\n", pRtlDispatchException);

j=0;
i=-1;

while(!j) {


i++;
for(; ((BYTE *)(pRtlDispatchException))[i] != 0xE8; i++)
continue;

if(((BYTE *)(pRtlDispatchException))[i+5] == 0xF6 && ((BYTE
*)(pRtlDispatchException))[i+6] == 0x05)
j++;

}

printf("call RtlpExecuteHandlerForException : %08X\n", pRtlD-
ispatchException+i);

pRtlpExecuteHandlerForException = pRtlDispatchException+i;
pRtlpExecuteHandlerForException += ( (DWORD *)(pRtlpEx-
ecuteHandlerForException+1) )[0] + 5;

printf("RtlpExecuteHandlerForException: %08X\n\n", pRtlpEx-
ecuteHandlerForException);
```

```
if((WORD)GetVersion() == 0x0005) {
// — Win 2K Begin —
printf("Windows 5.0\n");
pExecuteHandler = pRtlpExecuteHandlerForException+5;

pNewExecute2 = (DWORD)&NewExecute2;
pNewExecute2 -= pExecuteHandler+5;

WriteProcessMemory((HANDLE)-1,(LPVOID)(pExecuteHandler),
"\xE9",1,&lpWrByte);
WriteProcessMemory((HANDLE)-1,(LPVOID)(pExecuteHandler+1),
&pNewExecute2,4,&lpWrByte);


// — Win 2k END —
}

else if((WORD)GetVersion() == 0x0105) {
// — Win XP BEGIN —
printf("Windows 5.1\n");
pExecuteHandler = (pRtlpExecuteHandlerForException) + (((BYTE
*)(pRtlpExecuteHandlerForException))[6]) + 7;

printf("ExecuteHandler : %08X\n", pExecuteHandler);


for(i=0; ((BYTE *)(pExecuteHandler))[i] != 0xE8; i++)
continue;
pExecuteHandler = pExecuteHandler+i;

pNewExecute2 = (DWORD)&NewExecute2;
pNewExecute2 -= pExecuteHandler+5;

WriteProcessMemory((HANDLE)-1,(LPVOID)pExecuteHandler,
"\nxE8",1,&lpWrByte);
WriteProcessMemory((HANDLE)-1,(LPVOID)(pExecuteHandler+1),
&pNewExecute2,4,&lpWrByte);
// — Win XP END —
}

_asm {
```

```
push offset SEH
push dword ptr fs:[0]
mov dword ptr fs:[0], esp

xor eax, eax
mov [eax], eax

mov esp, dword ptr fs:[0]
pop dword ptr fs:[0]
add esp, 4

retn
SEH:
mov ecx,dword ptr ss:[esp+0xC]
add dword ptr ds:[ecx+0xB8],2
xor eax,eax
mov dword ptr ds:[ecx+0x4],eax
mov dword ptr ds:[ecx+0x8],eax
mov dword ptr ds:[ecx+0xC],eax
mov dword ptr ds:[ecx+0x10],eax
mov dword ptr ds:[ecx+0x14],eax
mov dword ptr ds:[ecx+0x18],155
retn

}

/*
pExecuteHandler += ( (DWORD *)(pExecuteHandler+1) )[0] +
5;
printf("ExecuteHandler2 : %08X
n
n", pExecuteHandler);

for(i=0; ; i++) {
if( ((BYTE *)(pExecuteHandler))[i] == 0xFF && ((BYTE *)(pEx-
ecuteHandler))[i+1] == 0xD1)
break;
}

printf("0x%08X : call ecx", pExecuteHandler+i);
/
```

```
return 1;
}
int __stdcall NewExecute2(DWORD ExceptionRecord, DWORD
EstablisherFrame, LPCONTEXT ContextRecord, DWORD Dis-
patcherContext,SEHandler Handler) {

DRx.Dr0 = ContextRecord->Dr0;
DRx.Dr1 = ContextRecord->Dr1;
DRx.Dr2 = ContextRecord->Dr2;
DRx.Dr3 = ContextRecord->Dr3;
DRx.Dr6 = ContextRecord->Dr6;
DRx.Dr7 = ContextRecord->Dr7;

printf("Eip: 0x%08X // SEH Handler : 0x%08X\n",ContextRecord-
>Eip, Handler);

Handler(ExceptionRecord, EstablisherFrame, ContextRecord, Dis-
patcherContext);

ContextRecord->Dr0 = DRx.Dr0;
ContextRecord->Dr1 = DRx.Dr1;
ContextRecord->Dr2 = DRx.Dr2;
ContextRecord->Dr3 = DRx.Dr3;
ContextRecord->Dr6 = DRx.Dr6;
ContextRecord->Dr7 = DRx.Dr7;

return 1;

}
```

3.4 Is it the same as in the Anti-Anti-Hardware plugin in ollydbg?

Yes, except the NewExecute(2) Handler is:

```
int __declspec(dllexport) __stdcall NewExecute2(DWORD Excep-
```

```
tionRecord, DWORD EstablisherFrame, LPCONTEXT Contex-
tRecord, DWORD DispatcherContext,SEHandler Handler) {
//DWORD FckHandler;

//_asm mov FckHandler, edx

DRx.Dr0 = ContextRecord->Dr0;
DRx.Dr1 = ContextRecord->Dr1;
DRx.Dr2 = ContextRecord->Dr2;
DRx.Dr3 = ContextRecord->Dr3;
DRx.Dr6 = ContextRecord->Dr6;
DRx.Dr7 = ContextRecord->Dr7;

if(if_changed) {
ContextRecord->Dr0 = _DRx.Dr0;
ContextRecord->Dr1 = _DRx.Dr1;
ContextRecord->Dr2 = _DRx.Dr2;
ContextRecord->Dr3 = _DRx.Dr3;
ContextRecord->Dr6 = _DRx.Dr6;
ContextRecord->Dr7 = _DRx.Dr7;
}

//printf("Eip: 0x%08X // SEH Handler : 0x%08X",ContextRecord-
>Eip, Handler);

/*
_asm {
push dword ptr ss:[ebp+0xC]
push FckHandler
push dword ptr fs:[0]
mov dword ptr fs:[0],esp
}
/
Handler(ExceptionRecord, EstablisherFrame, ContextRecord, Dis-
patcherContext);
/*
_asm {
mov esp,dword ptr fs:[0]
pop dword ptr fs:[0]
}
/
```

```
if((ContextRecord->Dr7 == 0x155) && (if_changed == 0)) {

_DRx.Dr0 = ContextRecord->Dr0;
_DRx.Dr1 = ContextRecord->Dr1;
_DRx.Dr2 = ContextRecord->Dr2;
_DRx.Dr3 = ContextRecord->Dr3;
_DRx.Dr6 = ContextRecord->Dr6;
_DRx.Dr7 = ContextRecord->Dr7;

ContextRecord->Dr0 = DRx.Dr0;
ContextRecord->Dr1 = DRx.Dr1;
ContextRecord->Dr2 = DRx.Dr2;
ContextRecord->Dr3 = DRx.Dr3;
ContextRecord->Dr6 = DRx.Dr6;
ContextRecord->Dr7 = DRx.Dr7;

if_changed = 1;

} else if (if_changed) {

ContextRecord->Dr0 = DRx.Dr0;
ContextRecord->Dr1 = DRx.Dr1;
ContextRecord->Dr2 = DRx.Dr2;
ContextRecord->Dr3 = DRx.Dr3;
ContextRecord->Dr6 = DRx.Dr6;
ContextRecord->Dr7 = DRx.Dr7;

} /* else {

if(ContextRecord->Dr0 != DRx.Dr0)
//olly_add_to_list(0,__ERROR__,"==> The Dr0 have been changed
!");

if (ContextRecord->Dr1 != DRx.Dr1)
//olly_add_to_list(0,__ERROR__,"==> The Dr1 have been changed
!");

if (ContextRecord->Dr2 != DRx.Dr2)
//olly_add_to_list(0,__ERROR__,"==> The Dr2 have been changed
!");
```

```
if (ContextRecord->Dr3 != DRx.Dr3)
//olly_add_to_list(0,__ERROR__,"==> The Dr3 have been changed
!");

if (ContextRecord->Dr6 != DRx.Dr6)
//olly_add_to_list(0,__ERROR__,"==> The Dr6 have been changed
!");

if (ContextRecord->Dr7 != DRx.Dr7)
//olly_add_to_list(0,__ERROR__,"==> The Dr7 have been changed
!");

} */

return 0;
}
```

Chapter 4

Links

An association that has developed standardized methods of citing sources for research.

[1] Intel 80386 Programmer's Reference 1986 - Chapter 12: "Debugging" ¡http://www.online.ee/ andre/i80386/Chap12.html¿
[2] Microsoft Systems Journal - Matt Pietrek - A Crash Course on the Depths of Win32 Structured Exception Handling - ¡http://www.microsoft.com/msj/0197/exception/exceptio
[3] FRET WebSite ¡http://www.binary-reverser.org/spip/¿
[4] Mattwood ¡www.pastapolis.info¿

Notice of this Century *Get The Pasta Power is the Best ! Believe Me !*