

Project Vesper

얘기는 별로 없어요...

MS 에게 속았어요 >.<

노용환 (a.k.a. somma)
fixbrain@gmail.com

저는...



->음<-
by somma

윈도우 쪼물딱거리기
(<http://somma.egloos.com>)

<https://github.com/somma/vesper>

VMCraft



네이버 책

다른 사이드를 보시려면 여기를 누르세요.



32세, 32명 만들기 **베리미러보기**
노출권 개 | 국립중앙경제연구소 | 2004.02.25

★★★★☆ 7.55 | 네티즌리뷰[14건]
소개 전세금 2천8백만원으로 시작하여 30대 중반에 노후준비를 끝낸 저자의 노하우가 담겨있다. 저자는 '불고기 같은 법은 어부에게 배우고 부동산 투자법은 전자...'
연관도서 **총서(돈 앞에 당당한 경제자유인 프로...)**



바닥할 때 거저먹는 재테크 **베리미러보기**
노출권 개 | 국일출판사 | 2004.07.25

★★★★☆ 6.0 | 네티즌리뷰[6건] | 도서구매 42,000원 → 9,480원(-21%)
소개 28세에 전세금 2,800만 원으로 결혼해서 1년 만에 내 집 마련에 성공한 재테크 컨설턴트 노용환의 재테크 전문 서적, 새로운 시각과 분석을 보여주고, 재테크 정보를...
연관도서 **총서(돈 앞에 당당한 경제자유인 프로...)**



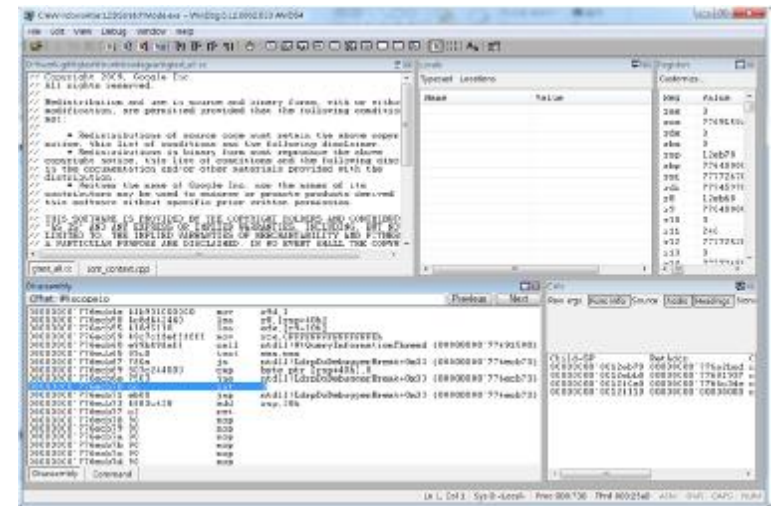
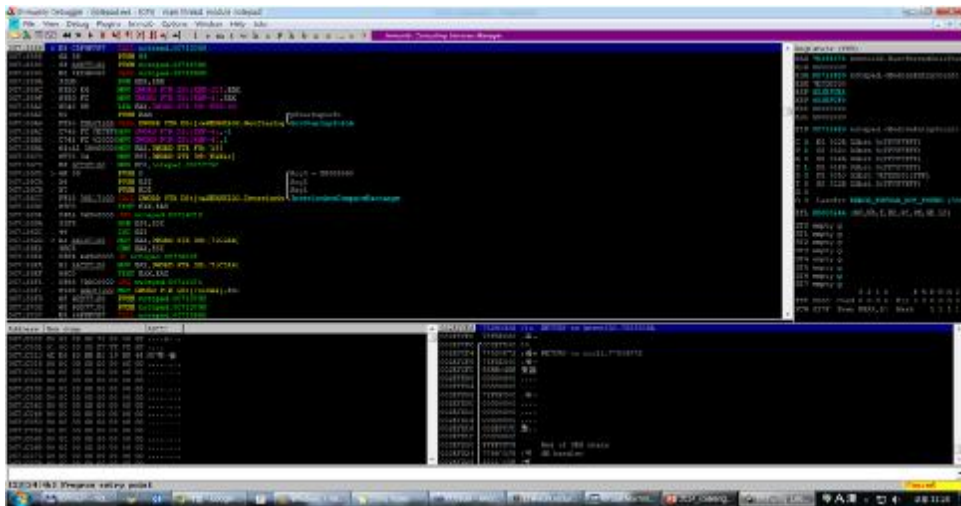
나는 펀드 투자를 5,000원 벌었다 **(새로 들어 갈아엎) 펀드 투자 테크닉**
노출권 개 | 국일미디어(국일출판사) | 2005.12.19

★★★★☆ 7.0 | 네티즌리뷰[11건] | 도서구매 40,000원 → 7,500원(-25%)
소개 <32세, 32명 만들기>, <바닥할 때 거저먹는 재테크>로 유명한 '노용환(드레곤)' 컨설턴트의 '펀드 투자기다'. 저자는 '펀드를 아는 것'과 '펀드에 투자해서 돈을 버는 것'은 별개라고 말한다. 이 책에는 노 소장이 겪은 다종다양한 경험과 혼자 깨달은 노하우들이 오롯이 녹아 있다.

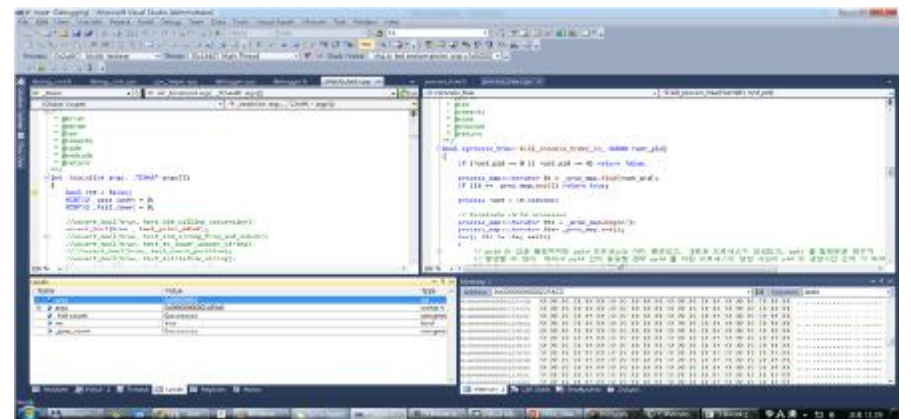
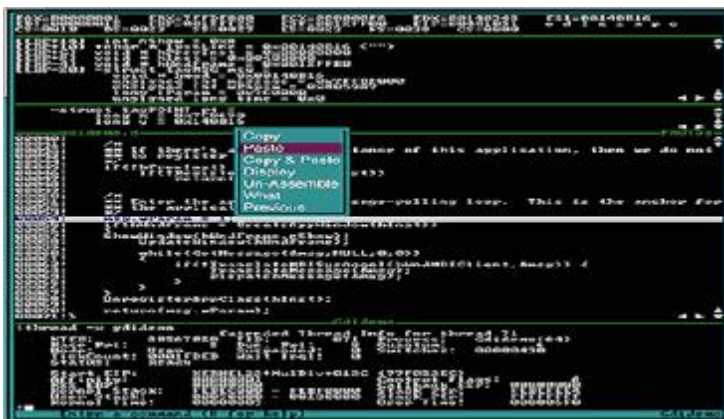


차세대 보안리더
양성 프로그램

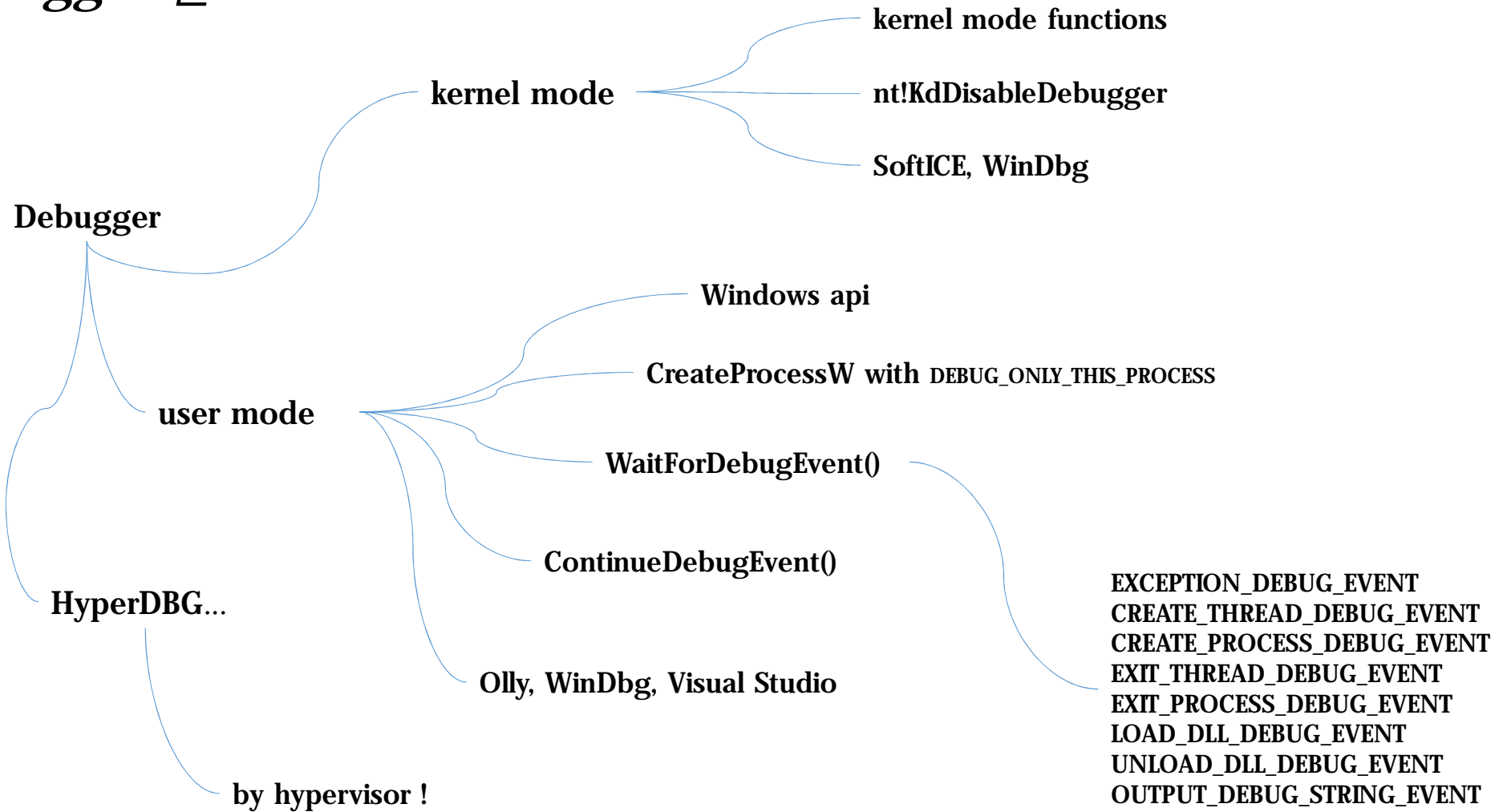




Debugger를 만들면서 겪었던 일을 이야기 하고자 합니다.
 + windows kernel 분석 방법도...
 + Vesper project 얘기 짝~꿈...



Debugger ≡



Vesper 는 *branch tracing* 에 특화된 *Debugger* 를 만들고자 해요...

네, 이미 많이 있습니다.
process stalker 도 있고,
olly 에서도 다 될 겠요?

그냥 하는...

Windows 살펴보기 - CONTEXT structure

```
GetThreadContext(current_thread, &context);
```

```
...
```

```
context.EFlags |= TRAP_FLAG; // set single step
```

```
...
```

```
SetThreadContext(current_thread, &context);
```

```
...
```

```
typedef struct DECLSPEC_ALIGN(16) _CONTEXT {  
    ...  
    WORD    SegCs;  
    ...  
    DWORD  EFlags;  
    ...  
    DWORD64 Dr6;  
    DWORD64 Dr7;  
    ...  
    DWORD64 DebugControl;  
    DWORD64 LastBranchToRip;  
    DWORD64 LastBranchFromRip;  
    DWORD64 LastExceptionToRip;  
    DWORD64 LastExceptionFromRip;  
} CONTEXT, *PCONTEXT;
```

LBR Stack 같은데?

윈도우 커널에 이미 구현되어있나?

어떻게 쓰는 거지?

Windows 살펴보기 – SetThreadContext

복잡해 보이는데... 하지 말까?..

```
1: kd> uf /c /D nt!NtSetContextThread
nt!NtSetContextThread (ffff800`02f2d760)
  nt!NtSetContextThread+0x43 (ffff800`02f2d7a3):
    call to nt!ObReferenceObjectByHandle (ffff800`02fb19a0)
  nt!NtSetContextThread+0x63 (ffff800`02f2d7c3):
    call to nt!PsSetContextThread (ffff800`02f2d740)
  nt!NtSetContextThread+0x74 (ffff800`02f2d7d4):
    call to nt!ObfDereferenceObject (ffff800`02ce1d80)
```

```
1: kd> uf /c /D 0xffff800`02f2d740
nt!PsSetContextThread (ffff800`02f2d740)
  nt!PsSetContextThread+0xc (ffff800`02f2d74c):
    call to nt!PspSetContextThreadInternal (ffff800`02fca4b4)
```

```
1: kd> uf /c /D 0xffff800`02fca4b4
nt!PspSetContextThreadInternal (ffff800`02fca4b4)
  ... 생략 ...
  call to nt!KeInsertQueueApc (ffff800`02cb9624)
  ... 생략 ...
```

Windows (좀 더 깊이) 살펴보기 - KiDebugtrapOrFault

```
INIT:0000000140555120 KiInterruptInitTable dq 0
INIT:0000000140555128 off_140555128 dq offset KiDivideErrorFault
INIT:0000000140555128 ; DATA XREF: KiInitializeBootStructures+119f0
INIT:0000000140555130 dq 1
INIT:0000000140555138 dq offset KiDebugTrapOrFault
INIT:0000000140555140 dq 30002h
INIT:0000000140555148 dq offset KiNmiInterruptStart
INIT:0000000140555150 dq 303h
INIT:0000000140555158 dq offset KiBreakpointTrap
INIT:0000000140555160 dq 304h
```



```
void hand_ray_KiDebugTrapOrFault()
{
    ...
    do
    {
        if (2 != gs:[4d4a]) break;
        if (0 == (dr7 && 0x200)) break;
        if (0 == (dr7 && 0x100)) break;

        if (0 == ntoskrnl_KiLastBranchTOMSR) break;
        local_KiLastBranchTOMSR_value = read_msr(ntoskrnl_KiLastBranchTOMSR);

        local_KiLstBranchFromBaseMSR_value = read_msr(ntoskrnl_KiLstBranchFromBaseMSR);
        ...
        생략
        ....
    } while(false);
    ....
}
```

```
.text:0000000140072B27 test byte ptr gs:404Ah, 2
.text:0000000140072B30 jz loc_140072BC7
.text:0000000140072B36 test [rbp+0E8h+arg_0], 1
.text:0000000140072B3D jnz short loc_140072B8E
.text:0000000140072B3F mov rax, dr7
.text:0000000140072B42 test ax, 200h
.text:0000000140072B46 jz short loc_140072BC7
.text:0000000140072B48 test ax, 100h
.text:0000000140072B4C jz short loc_140072BC7
.text:0000000140072B4E mov r8d, cs:KiLastBranchTOMSR
.text:0000000140072B55 or r8d, r8d
.text:0000000140072B58 jz short loc_140072B62
.text:0000000140072B5A mov ecx, r8d
.text:0000000140072B5D rdmsr
.text:0000000140072B5F mov r8d, eax
.text:0000000140072B62 loc_140072B62: ; CODE XREF: KiDebugTrapOrFault+118fj
.text:0000000140072B62 mov ecx, cs:KiLastBranchFromBaseMSR
.text:0000000140072B68 add ecx, r8d
.text:0000000140072B6B rdmsr
```

역시 hand-ray 가 짱이죠!

Windows (조금 더~더 깊이) 살펴보기 - KiSaveDebugRegisterState

```
1: kd> uf nt!KiSaveDebugRegisterState
nt!KiSaveDebugRegisterState:
...

nt!KiSaveDebugRegisterState+0xcfc:
fffff800`02cd913f 498b81e0010000 mov     rax,qword ptr [r9+1E0h]
fffff800`02cd9146 498b91e8010000 mov     rdx,qword ptr [r9+1E8h]
fffff800`02cd914d 0f23c0    mov     dr0,rax
fffff800`02cd9150 0f23ca    mov     dr1,rdx
fffff800`02cd9153 498b81f0010000 mov     rax,qword ptr [r9+1F0h]
fffff800`02cd915a 498b91f8010000 mov     rdx,qword ptr [r9+1F8h]
fffff800`02cd9161 0f23d0    mov     dr2,rax
fffff800`02cd9164 0f23da    mov     dr3,rdx
fffff800`02cd9167 498b9108020000 mov     rdx,qword ptr [r9+208h]
fffff800`02cd916e 33c0     xor     eax,eax
fffff800`02cd9170 0f23f0    mov     dr6,rax
fffff800`02cd9173 0f23fa    mov     dr7,rdx
fffff800`02cd9176 65f604254a4d000002 test    byte ptr gs:[404Ah],2
...
fffff800`02cd9181 66f7c20002 test    dx,200h
...
fffff800`02cd9188 83c802    or     eax,2
...
fffff800`02cd918b 66f7c20001 test    dx,100h
...
fffff800`02cd9192 83c801    or     eax,1
...
nt!KiSaveDebugRegisterState+0x129:
fffff800`02cd9199 448bc0    mov     r8d,eax
fffff800`02cd919c b9d9010000 mov     ecx,1D9h
fffff800`02cd91a1 0f32     rdmsr
fffff800`02cd91a3 83e0fc    and     eax,0FFFFFFCh
fffff800`02cd91a6 410bc0    or     eax,r8d
fffff800`02cd91a9 0f30     wrmsr
```



```
void hand_ray()
{
    dr0 = some_struct.dr0;
    dr1 = some_struct.dr1;
    ...
    dr6 = some_struct.dr6;
    dr7 = some_struct.dr7;

    if (gs:[4d4a] == 2)
    {
        uint32_t msr_value_new = 0;

        if (dr7 & 0x200)
        {
            msr_value_new |= 2;
        }

        if (DR7 & 0x100)
        {
            msr_value_new |= 1;
        }

        msr_value = read_msr(0x01d9);

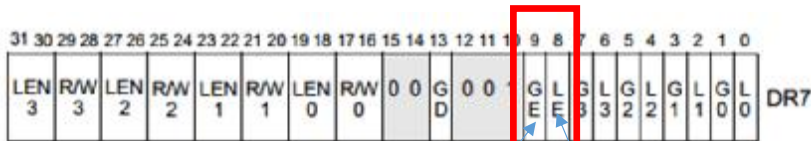
        // 최 하위 두 비트 클리어
        msr_value &= 0xFFFFF0C;

        // DR7 값에 따라서 설정 한 두 비트 값 설정
        msr_value |= msr_value_new;

        // MSR 0x01D9 값 설정
        write_msr(0x01d9, msr_value);
    }
}
```

정리 - DR7 과 IA32_DEBUGCTL MSR 간의 관계

LE and GE (local and global exact breakpoint enable) flags (bits 8, 9) — This feature is not supported in the P6 family processors, later IA-32 processors, and Intel 64 processors. When set, these flags cause the processor to detect the exact instruction that caused a data breakpoint condition. For backward and forward compatibility with other Intel processors, we recommend that the LE and GE flags be set to 1 if exact breakpoints are required.



DR7 레지스터

LE (0x100)
GE (0x200)

IA32_DEBUGCTL MSR

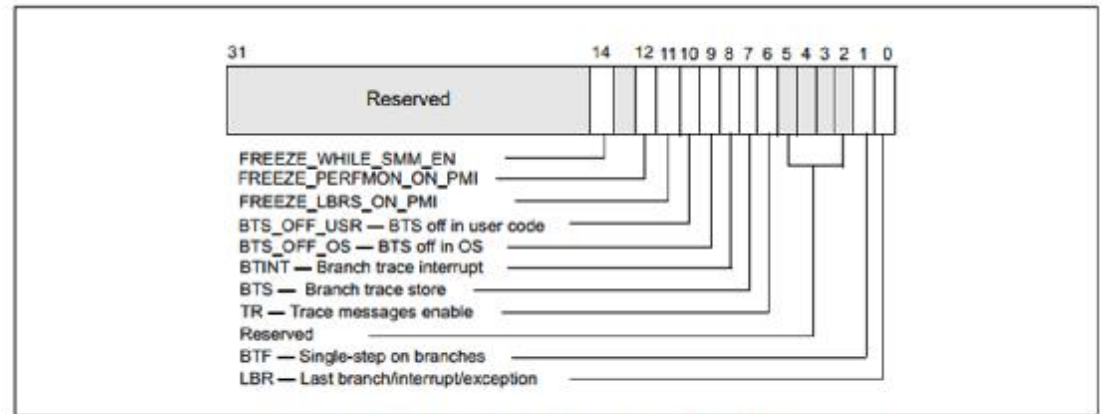


Figure 17-3. IA32_DEBUGCTL MSR for Processors based on Intel Core microarchitecture

LBR (bit 0) - 마지막 branch, exception, interrupt 를 LBR stack 에 저장

BTF (bit 1) - **Single Step On Branch**

그렇다면...

#1 enable IA32_DEBUGCTL MSR.BTF

```
bool write_IA32_DEBUGCTL_msr(_In_ DWORD msr_value)
{
    if (true != set_privilege(SE_DEBUG_NAME, true))
    {
        _tprintf(_T("set_privilege() failed \n"));
        return false;
    }

    HMODULE nt = GetModuleHandleW(L"ntdll.dll");
    fnNtSystemDebugControl func = (fnNtSystemDebugControl)
        GetProcAddress(
            nt,
            "NtSystemDebugControl"
        );

    //> write msr
    ULONG ReturnLength = 0;
    SYSDBG_MSR msr = { IA32_DEBUGCTL_MSR, 0x0000000000000000 };
    msr.Data |= MSR_IA32_DEBUGCTL_BTF;
    NTSTATUS status = func(
        SysDbgWriteMsr,
        &msr,
        sizeof(msr),
        &msr,
        sizeof(msr),
        &ReturnLength
    );

    return true;
}
```

#2 DR7.GE = 1 & ELFLAGS.TF = 1

```
bool set_lbr_enable()
{
    #define TF_BIT            0x100
    #define DR7_TRACE_BRANCH 0x200
    #define DR7_LAST_BRANCH  0x100

    CONTEXT context = {0};
    context.ContextFlags = CONTEXT_ALL;
    if (TRUE != GetThreadContext(GetCurrentThread(), &context))
    {
        return false;
    }

    context.EFlags |= TF_BIT;
    context.Dr7 |= (DR7_TRACE_BRANCH);

    if (TRUE != SetThreadContext(GetCurrentThread(), &context))
    {
        return false;
    }

    return true;
}
```

간단하구만!!!



다시 보기 - KiDebugtrapOrFault

```
.text:0000000140072B27
.text:0000000140072B30
.text:0000000140072B36
.text:0000000140072B3D
.text:0000000140072B3F
.text:0000000140072B42
.text:0000000140072B46
.text:0000000140072B48
.text:0000000140072B4C
.text:0000000140072B4E
.text:0000000140072B55
.text:0000000140072B58
.text:0000000140072B5A
.text:0000000140072B5D
.text:0000000140072B5F
.text:0000000140072B62
loc_140072B62:
.text:0000000140072B62
.text:0000000140072B68
.text:0000000140072B6B

test     byte ptr gs:4D4Ah, 2
jz       loc_140072B67
test     byte ptr gs:4D4Ah, 2
jnz      loc_140072B67
mov     eax, 0
test     byte ptr gs:4D4Ah, 2
jz       loc_140072B67
test     byte ptr gs:4D4Ah, 2
jz       loc_140072B67
mov     eax, 0
or      eax, 0
jz       loc_140072B67
mov     eax, 0
rdmsr   eax, 0
mov     eax, 0
if (2 != gs:[4d4a]) break;
if (0 == (dr7 && 0x200)) break;
if (0 == (dr7 && 0x100)) break;
if (0 == ntoskrnl_KiLastBranchTOMSR) break;
local_KiLastBranchTOMSR_value = read_msr(ntoskrnl_KiLastBranchTOMSR);
local_KiLstBranchFromBaseMSR_value = read_msr(ntoskrnl_KiLstBranchFromBaseMSR);
...
생략
....
} while(false);
....
}
```

GS:[4D4A]의 값이 2가 아니면 **BTF**, **LBR** 기능이 동작하지 않습니다. 저게 대체 뭘까요?

GS:[4D4A] 가 뭐길래...

```
1: kd> r gs
gs=002b
1: kd> rdmsr 0xc0000101 ; IA32_GS_BASE msr 에 GS 의 주소가 있습니다.
msr[c0000101] = ffffffff800009ea000

1: kd> db ffffffff800009ea000 + 0x4d4a L1
fffffff800009eed4a 19

1: kd> db gs:[4d4a] L1 ; 간단히 이렇게 해도 됩니다.
002b:0000000000004d4a 19
```

0x19 != 2 이므로

안 되는 것이 맞는 것 같긴 하지만,
CPU 에서 **LBR, BTF** 를 지원하는데...?

분석 전략 - GS:[4d4a]

reboot system

set memory break point at GS:[4D4A]

사실은...

bp 를 설치한 채로 부팅을 계속하면
BSOD 가 발생...

```
; 시스템 부팅 후 initial breakpoint 시절
; gs 세그먼트의 베이스 주소들 알아냅니다.
kd> rdmsr 0xc0000101
msr[c0000101] = ffffffff80002e0cd00

; 현재 0x19 가 들어있습니다.
kd> db ffffffff80002e0cd00 + 4d4a L1
fffffff80002e11a4a 19

; gs:[4d4a] 에 1바이트 쓰기 브레이크 포인트를 걸어줍니다.
kd> ba w1 ffffffff80002e0cd00 + 4d4a
kd> bl
0 e ffffffff80002e11a4a w 1 0001 (0001) nt!KiInitialPCR+0x4d4a

; 브레이크 포인트를 사용할 일도 없습니다.
; gs:[0] 은 nt!KiInitialPCR 의 포인터입니다.
; 참고로 x86 에서는 FS:[0] 가 nt!KiInitialPCR 포인터였습니다.
kd> ln ffffffff80002e0cd00 + 4d4a
Browse module
Clear breakpoint 0

(fffdff80002e0cd00) nt!KiInitialPCR+0x4d4a | (fffff80002e15b40) nt!KiCacheFlushTimeStamp
```

GS:[4D4A] 좀 더 자세히...

```
1: kd> dt _KPCR
ntdll!_KPCR
+0x000 NtTib      : _NT_TIB
...
+0x180 Prcb       : _KPRCB
```



```
0: kd> dt _KPRCB (fffff800`02dfad00 + 0x180)
ntdll!_KPRCB
+0x000 MxCsr      : 0x1f80
...
+0x5f0 CpuType    : 6  ''
+0x5f1 CpuID      : 1  ''
+0x5f2 CpuStep    : 0x3a09
+0x5f2 CpuStepping : 0x9  ''
+0x5f3 CpuModel   : 0x3a  ':'
...
+0x4bb8 VendorString : [13] "GenuineIntel" ; 뭔가 계대로 들어있습니다.
...
+0x4bc8 FeatureBits : 0x21193ffe ; 애가 범인입니다.
...
+0x4c80 RequestMailbox : [1] _REQUEST_MAILBOX
```

```
0: kd> db (fffff800`02dfad00 + 0x180 + 0x4bc8) L4
fffff800`02dffa48 fe 3 19 21 .?!
```

```
1: kd> dt _KPRCB
ntdll!_KPRCB
+0x000 MxCsr      : Uint4B
...
+0x5f0 CpuType    : Char
+0x5f1 CpuID      : Char
+0x5f2 CpuStep    : Uint2B
+0x5f2 CpuStepping : UChar
+0x5f3 CpuModel   : UChar
+0x5f4 MHz        : Uint4B
...
+0x4bb8 VendorString : [13] UChar
+0x4bc5 PrcbPad10    : [3] UChar
+0x4bc8 FeatureBits : Uint4B ; !! 여기입니다 !!
+0x4bd0 UpdateSignature : _LARGE_INTEGER
...
+0x4c80 RequestMailbox : [1] _REQUEST_MAILBOX
```

실제 주소로 매핑 해보니...

- 4바이트 변수의 3번째 바이트
- 비트 필드 스트럭처 일 것이다!
- 각 비트가 CPU의 특정 기능을 나타낼 것으로 예상...
- 대체 어떤 기능일까?

KPCR.Prcb.FeatureBits ...

이걸... 계속 해야 하나?...

분석 전략

- # **KiInitialPCR** 포인터이므로 **KiInitialXXXX** 류의 함수들이 접근할 것
- # **KiInitialXXX** 함수들 몽땅 읽어보면 되겠습니다 :-)

```
0: kd> x nt!KiInitial*
...
fffff800`02fff9d0 nt!KiInitializeProcessorState = <no type information>
...
fffff800`02ec29f0 nt!KiInitializePrcbContext = <no type information>
fffff800`02ecef00 nt!KiInitializeBootStructures = <no type information>
...
fffff800`02dfad00 nt!KiInitialPCR = <no type information>  << 애는 확실히 아님
...
```

```
0: kd> uf /c /D nt!KiInitializeBootStructures
nt!KiInitializeBootStructures (fffff800`02ecef00)
  nt!KiInitializeBootStructures+0x217 (fffff800`02ecf1e7):
    call to hal!HalInitializeBios (fffff800`03200318)
  nt!KiInitializeBootStructures+0x221 (fffff800`02ecf1f1):
    call to nt!InbvDriverInitialize (fffff800`0318c200)
  nt!KiInitializeBootStructures+0x23e (fffff800`02ecf20e):
    call to nt!strstr (fffff800`02ccd138)
  nt!KiInitializeBootStructures+0x2be (fffff800`02ecf28e):
    call to nt!Hv1InitSystem (fffff800`02ecee80)
  nt!KiInitializeBootStructures+0x2c6 (fffff800`02ecf296):
    call to nt!KiSetFeatureBits (fffff800`02ebf450)
  nt!KiInitializeBootStructures+0x2d0 (fffff800`02ecf2a0):
    call to nt!KiInitSpinLocks (fffff800`02ec3bb0)
  ...
<< 이름만 봐도 확실합니다.
```


nt!KiSetFeatureBits ...

이걸...계속 해야 하나?...그만할까...

분석 전략

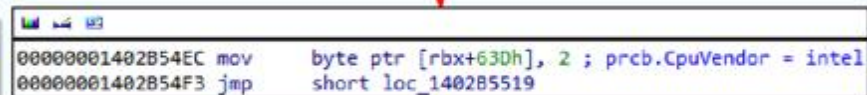
왕도는 없다. 집중해서 보는 것 말고는...

`cpuid` 명령어를 많이 호출할 것이고, `cpu vender`, `cpu type`, `cpu family` 등의 값을 이용해 특정 기능 사용 여부 등을 저장하는 기능일 것이다.

```
00000000140285470 call    KiCpuId          ; eax = 0
00000000140285470                ; cpuid
00000000140285470                ; ---
00000000140285470                ; EAX : Maximum Input Value for Basic CPUID Information (see Table 3-18)
00000000140285470                ; Intel Core i7 Processor
00000000140285470                ; - 00H (basic information)
00000000140285470                ; - 80000000H (extended function information)   시작부터 CPUID...
00000000140285470                ; EBX : "Genu"
00000000140285470                ; ECX : "ntel"
00000000140285470                ; EDX : "ineI"
00000000140285470                ; ---
```



```
000000001402854CA mov    [rbx+63Dh], r15b ; prcb.CpuVender = amd
000000001402854D1 jmp    short loc_140285519
```

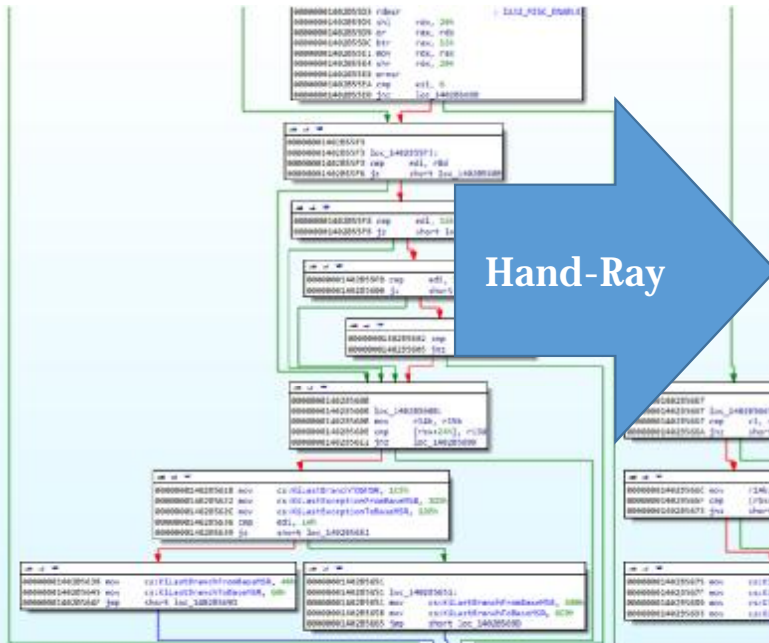


```
000000001402854EC mov    byte ptr [rbx+63Dh], 2 ; prcb.CpuVendor = intel
000000001402854F3 jmp    short loc_140285519
```

CpuVender = 1 : AMD
CpuVender = 2 : Intel

nt!KiSetFeatureBits ...

이걸...계속 해야 하나?...그만할까...그만하고 싶다...



```

...
if (prcb.CpuVender == AMD)
{
    nt.KiLastBranchFromBaseMSR = 0x01db;
    nt.KiLastBranchFromBaseMSR = 0x01dc;
    nt.KiLastBranchFromBaseMSR = 0x01dd;
    nt.KiLastBranchFromBaseMSR = 0x01de;
}
else
{
    if (prcb.CpuId == 6)
    {
        if (
            prcb.CpuModel == 0x16 || // merom, conroe (Celeron™ Desktop, Celeron™ Mobile)
            prcb.CpuModel == 0x17 || // Yorkfield (Core™ 2 Quad, Core™ 2 Extreme, Xeon™ 3000)
            prcb.CpuModel == 0x1A // Bloomfield (Core™ i7 Extreme, Core™ i7, Xeon™ 3000)
        )
        {
            nt.KiLastBranchTOSMSR = 0x01C9;
            nt.KiLastExceptionFromBaseMSR = 0x01DD;
            ....
        }
    }
}
...

```

```

0: kd> dt _KPRCB (fffff800`02dfad00 + 0x180)
ntdll!_KPRCB
...
+0x5f0 CpuType : 6 ''
...
+0x5f3 CpuModel : 0x3a ''
...

```

IvyBridge, Core i7 - BTF, LBR 지원합니다!!!

intel cpu type & model <http://goo.gl/18nmW>

그래서...결론은...

MS 가 구현해 둔 **LBR/BTF** 는 쓸 수 없습니다.

AMD CPU 도 아니고, 제 **CPU** 는 최신이거든요.

의도된 버그일까요? 아님 그냥 사소한 실수일까요?
windows 8.1 도 동일하던데...

종다가 말았습니다.

결국 **Driver** 를 구현해야 겠군요...

Vesper 는 ... 드라이버를 만들었습니다.

아... 귀찮아...인증서도 없는데...

```
NTSTATUS enable_btf()
{
    MSR msr = {0};
    NTSTATUS status = read_msr(MSR_IA32_DEBUGCTL, &msr);
    if (TRUE != NT_SUCCESS(status))
    {
        log_err "read_msr(MSR_IA32_DEBUGCTL), status = 0x%08x", status
        return status;
    }

    if (MSR_IA32_DEBUGCTL_BTF != (msr.low & MSR_IA32_DEBUGCTL_BTF))
    {
        msr.low |= MSR_IA32_DEBUGCTL_BTF;
        status = write_msr(MSR_IA32_DEBUGCTL, &msr);
        if (TRUE != NT_SUCCESS(status))
        {
            log_err "write_msr(MSR_IA32_DEBUGCTL_BTF), status = 0x%08x"
            return status;
        }
    }

#ifdef DBG
    //read_msr(MSR_IA32_DEBUGCTL, &msr);
    //log_info "set msr low = 0x%08x, high = 0x%08x", msr.low, msr.
#endif
}

return STATUS_SUCCESS;
}

NTSTATUS disable_btf()
{
    MSR msr = {0};
    NTSTATUS status = read_msr(MSR_IA32_DEBUGCTL, &msr);
    if (TRUE != NT_SUCCESS(status))
    {
        log_err "read_msr(MSR_IA32_DEBUGCTL), status = 0x%08x", status log_end
        return status;
    }

    if (MSR_IA32_DEBUGCTL_BTF == (msr.low & MSR_IA32_DEBUGCTL_BTF))
    {
        msr.low &= ~MSR_IA32_DEBUGCTL_BTF;
        status = write_msr(MSR_IA32_DEBUGCTL, &msr);
        if (TRUE != NT_SUCCESS(status))
        {
            log_err "write_msr(MSR_IA32_DEBUGCTL_BTF), status = 0x%08x", status log_end
            return status;
        }
    }

#ifdef DBG
    read_msr(MSR_IA32_DEBUGCTL, &msr);
    log_info "set msr low = 0x%08x, high = 0x%08x", msr.low, msr.high log_end
#endif
}

return STATUS_SUCCESS;
}
```

https://github.com/somma/vesper/blob/master/vesper_support/vesper_support.cpp

VM Evasion

흠... 얻어걸렸네!?

```
void check_vm_by_msr()
{
    set_privilege(SE_DEBUG_NAME, true);

    HMODULE nt = GetModuleHandleW(L"ntdll.dll");
    fnNtSystemDebugControl func = (fnNtSystemDebugControl) GetProcAddress(nt, "NtSystemDebugControl");

    //> write msr
    ULONG ReturnLength = 0;
    SYSDBG_MSR msr = { IA32_DEBUGCTL_MSR, 0x0000000000000000 };
    msr.Data |= MSR_IA32_DEBUGCTL_BTF;

    NTSTATUS status = func(SysDbgWriteMsr, &msr, sizeof(msr), &msr, sizeof(msr), &ReturnLength);
    if (TRUE != NT_SUCCESS(status))
    {
        _tprintf(_T("SysDbgWriteMsr failed, status = 0x%08x \n"), status);
        return false;
    }

    //> read msr
    status = func(SysDbgReadMsr, &msr, sizeof(msr), &msr, sizeof(msr), &ReturnLength);
    if (TRUE != NT_SUCCESS(status))
    {
        _tprintf(_T("SysDbgReadMsr failed, status = 0x%08x \n"), status);
        return false;
    }

    //> check
    if (MSR_IA32_DEBUGCTL_BTF == (msr.Data & MSR_IA32_DEBUGCTL_BTF))
    {
        _tprintf(_T("real machine!!!\n"));
    }
    else
    {
        _tprintf(_T("virtual machine!!!\n"));
    }
}
```

그래서...

Vesper 는 VM 환경에서는 사용할 수 없어요...

IA32_DEBUGCTL 기능 지원하도록 하이퍼바이저를 고쳐봐야겠어요.

같이 하실 분??

성능이 나오겠냐...
아니한 만 못하겠네...

질문 있으신가요?

재미없는 얘기

들어주셔서 ...

감사합니다.

이제 다들 일어나세요. 집에 가야죠~
J