

Forum:[Hack Forums](#)

Thread: [\[SRC\] Intercepting system service calls on WoW64 Processes](#)

excr4sh - June 30, 2015, 11:38 pm

Hello, so I decide to share this project I wrote some time ago, basically what this code does is set a hook at wow64cpu!X86SwitchTo64BitMode in order to intercept all sistem services functions, the advantage of this method is that you can intercept all functions with a single hook. implement the code as a DLL project, so to use it you should inject the DLL in the target process, injector is included.

monitoring calls in firefox:

[img]http://s4.postimg.org/4f0ijfbml/firefox_hook.jpg[/img]

Note that this method can be avoided using native calls (64-bit version of ntdll) or direct syscall in any case normal applications do not.

MSVC project can be downloaded from my github:

<https://github.com/Excr4sh/Wow64SsvcIntrcp>

Quick look:

Code #include "Wow64Hook.h"

#include "LinkedList.h"

HANDLE CurrentProcessHandle;

HANDLE ProcessHeapHandle;

LIST_ENTRY WoW64ListHead;

PVOID JumpSwitchBaseAddress;

PWOW64_SYSCALL_INFO AllocWow64Entry(

 In PSTR FunctionName,

 In ULONG Index

)

{

 PWOW64_SYSCALL_INFO Entry;

 SIZE_T FunctionNameLen;

 PSTR FunctionNameAlloc;

 FunctionNameLen = strlen(FunctionName);

 if (!(FunctionNameLen > 0))

 {

 return NULL;

 }

 Entry = (PWOW64_SYSCALL_INFO)HeapAlloc(ProcessHeapHandle, HEAP_ZERO_MEMORY, sizeof(WOW64_SYSCALL_INFO));

 FunctionNameAlloc = (PSTR)HeapAlloc(ProcessHeapHandle, HEAP_ZERO_MEMORY, FunctionNameLen + sizeof(CHAR));

 if (!(Entry && FunctionNameAlloc))

 {

```

    if (Entry)
    {
        HeapFree(ProcessHeapHandle, 0, Entry);
    }
    if (FunctionNameAlloc)
    {
        HeapFree(ProcessHeapHandle, 0, FunctionNameAlloc);
    }

    return NULL;
}

RtlCopyMemory(FunctionNameAlloc, FunctionName, FunctionNameLen);
Entry->FunctionName = FunctionNameAlloc;
Entry->Index = Index;

return Entry;
}

```

```

BOOL AddWoW64FunctionEntry(
    _In_ PSTR FunctionName,
    _In_ PVOID FunctionAddress
)
{
    PBYTE BytesAddress;
    ULONG Index;
    PWOW64_SYSCALL_INFO NewWow64Entry;

```

/*

```

ZwCreateFile  B8 52000000  MOV EAX, <Index>
              33C9        XOR ECX,ECX
              8D5424 04    LEA EDX,DWORD PTR SS:[ESP+0x4]
              64:FF15 C000000>CALL NEAR DWORD PTR FS:[0xC0]
              83C4 04     ADD ESP,0x4
              C2 2C00     RETN 0x2C

```

Fast check, this can be improved by using a disassembler

*/

```

BytesAddress = (PBYTE)FunctionAddress;

```

```

if (BytesAddress[0] == 0xB8 &&    // MOV
    BytesAddress[7] == 0x8D &&    // LEA
    BytesAddress[0xB] == 0x64 &&  // CALL
    BytesAddress[0xC] == 0xFF &&
    BytesAddress[0xD] == 0x15 &&
    BytesAddress[0x12] == 0x83 && // ADD
    (BytesAddress[0x15] == 0xC2 || BytesAddress[0x15] == 0xC3) // RET
)
{
    Index = *((PULONG)&BytesAddress[1]);
    NewWow64Entry = AllocWow64Entry(FunctionName, Index);
}

```

```

    if (NewWow64Entry)
    {
        INSERT_TAIL_LIST(&WoW64ListHead, &NewWow64Entry->ListEntry);
        return TRUE;
    }
}

return FALSE;
}

BOOL EnumWow64SystemFunctions()
{
    HMODULE NtdllBaseAddress;
    PIMAGE_DOS_HEADER DOSHeader;
    PIMAGE_NT_HEADERS NTHeaders;
    PIMAGE_EXPORT_DIRECTORY ExportDirectory;
    PULONG NameTable;
    PUSHORT OrdinalTable;
    PULONG AddressTable;
    USHORT Ordinal;
    PSTR SymbolName;
    PVOID SymbolAddress;

    NtdllBaseAddress = GetModuleHandle(_T("ntdll"));
    DOSHeader = (PIMAGE_DOS_HEADER)NtdllBaseAddress;
    NTHeaders = (PIMAGE_NT_HEADERS)((ULONG_PTR)NtdllBaseAddress +
DOSHeader->e_lfanew);

    ExportDirectory = (PIMAGE_EXPORT_DIRECTORY)((ULONG_PTR)NtdllBaseAddress +
NTHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddre
ss);
    NameTable = (PULONG)((ULONG_PTR)NtdllBaseAddress +
ExportDirectory->AddressOfNames);
    OrdinalTable = (PUSHORT)((ULONG_PTR)NtdllBaseAddress +
ExportDirectory->AddressOfNameOrdinals);
    AddressTable = (PULONG)((ULONG_PTR)NtdllBaseAddress +
ExportDirectory->AddressOfFunctions);

    for (DWORD n = 0; n < ExportDirectory->NumberOfNames; n++)
    {
        Ordinal = OrdinalTable[n];
        if (Ordinal &&
            (!(Ordinal >= ExportDirectory->NumberOfFunctions)))
        {
            SymbolName = (PSTR)((ULONG_PTR)NtdllBaseAddress + NameTable[n]);
            SymbolAddress = (PVOID)((ULONG_PTR)NtdllBaseAddress + AddressTable[Ordinal]);
            AddWoW64FunctionEntry(SymbolName, SymbolAddress);
        }
    }

    return TRUE;
}

```

```

PWSTR FormatCallString(
    _In_ _Printf_format_string_ PWSTR Format,
    ...
)
{
    va_list argptr;
    va_start(argptr, Format);
    return FormatCallStringEx(Format, argptr);
}

```

```

PWSTR FormatCallStringEx(
    _In_ _Printf_format_string_ PWSTR Format,
    _In_ va_list ArgPtr
)
{
    PWSTR String;
    ULONG Length;
    SIZE_T BufferSize;

    Length = _vscwprintf(Format, ArgPtr);
    if (Length == -1)
    {
        return NULL;
    }

    BufferSize = (Length * sizeof(WCHAR)) + sizeof(WCHAR);
    String = (PWSTR)HeapAlloc(ProcessHeapHandle, HEAP_ZERO_MEMORY, BufferSize);

    if (String)
    {
        _vsnwprintf(String, Length, Format, ArgPtr);
    }

    return String;
}

```

```

PWSTR fMultiCharToUnicode(_In_ PSTR Ansi)
{
    PWSTR Pointer;
    ULONG Length;

    Length = MultiByteToWideChar(CP_UTF8, 0, Ansi, -1, NULL, 0);
    Pointer = (PWSTR)HeapAlloc(ProcessHeapHandle, HEAP_ZERO_MEMORY, Length *
sizeof(WCHAR));
    if (Pointer)
    {
        if (!MultiByteToWideChar(CP_UTF8, 0, Ansi, -1, Pointer, Length))
        {
            HeapFree(ProcessHeapHandle, 0, Pointer);
            Pointer = NULL;
        }
    }
}

```

```

    return Pointer;
}

VOID __stdcall IndexLookupPrint(_In_ DWORD Index)
{
    PLIST_ENTRY List;
    PWOW64_SYSCALL_INFO Entry;
    PWSTR UnicodeName;
    PWSTR DebugString;

    List = WoW64ListHead.Flink;

    // synchronization is unnecessary in this code
    while (List != &WoW64ListHead)
    {
        Entry = CONTAINING_RECORD(List, WOW64_SYSCALL_INFO, ListEntry);
        if (Entry->Index == Index)
        {
            UnicodeName = fMultiCharToUnicode(Entry->FunctionName);
            if (UnicodeName)
            {
                DebugString = FormatCallString(_T("Call:%s (0x%X)", UnicodeName, Index);
                if (DebugString)
                {
                    OutputDebugStringW(DebugString);
                    HeapFree(ProcessHeapHandle, 0, DebugString);
                }

                HeapFree(ProcessHeapHandle, 0, UnicodeName);
            }

            break;
        }

        List = List->Flink;
    }
}

VOID __declspec(naked) X86SwitchTo64BitModeProxy()
{
    __asm
    {
        pushad
        mov edi, JmpSwitchBaseAddress
        mov esi, dword ptr fs:[0xC0]
        mov dword ptr fs:[0xC0], edi
        push eax
        call IndexLookupPrint
        mov dword ptr fs:[0xC0], esi
        popad
        jmp JmpSwitchBaseAddress;
    }
}

```

hezboula123 - June 30, 2015, 11:43 pm

Nice, clean. I like it.

Good job.

Ironically intercepting sys calls on wow64 is more straight forward than on x86. KiFastSystemCall isn't large enough for a five byte relative jump instruction.

I like the idea of not using a LDE for Nt* Functions. You can always guarantee the patch size.

I still cringe at Microsoft generic linked list implementation. oh well.

excr4sh - June 30, 2015, 11:52 pm

(Jun 30, 2015 11:43 PM)hezboula123 Wrote: [\[CITE\]](#) Nice, clean. I like it.

Good job.

Ironically intercepting sys calls on wow64 is more straight forward than on x86. KiFastSystemCall isn't large enough for a five byte relative jump instruction.

I like the idea of not using a LDE for Nt* Functions. You can always guarantee the patch size.that's true, it requires less than 5-bytes long so you can make it work but that's even possible. I may add it to the project so it work in both 32 and 64b-bit Windows.

hezboula123 - June 30, 2015, 11:55 pm

(Jun 30, 2015 11:52 PM)FordCI Wrote: [\[CITE\]](#) that's true, it requires less than 5-bytes long so you can make it work but that's even possible. I may add it to the project so it work in both 32 and 64b-bit Windows.

Definitely possible. backwards short jumps that lead to relative 4 byte jumps would be a solution.

```
CONTAINING_RECORD(List, WOW64_SYSCALL_INFO, ListEntry);
```

Let me guess what this does, considering type of listentry exists in type of wow64_syscall_info this macro finds the corresponding 'list' pointer along with the other data types defined in the type of 'wow64_syscall_info' structure in memory then returns the base (type of wow64_syscall_info)?

I'm guessing since the address of the list entry is +8 of the structure you can just subtract 8 from the pointer, it's essentially what the macro is doing i believe.

Edit:

Just looked up the macro, looks like i was right.

```
(type *)((char*)(address) - (unsigned long)(amp((type *)0)->field)))
```

RISC - July 1, 2015, 1:02 am

(Jun 30, 2015 11:43 PM)hezboula123 Wrote: [\[CITE\]](#) Nice, clean. I like it.

Good job.

Ironically intercepting sys calls on wow64 is more straight forward than on x86. KiFastSystemCall isn't large enough for a five byte relative jump instruction.

I like the idea of not using a LDE for Nt* Functions. You can always guarantee the patch size.

I still cringe at Microsoft generic linked list implementation. oh well.

You can make a two byte relative jump to KiIntSystemCall and perform the five byte patch from there. It's more straight forward than X86SwitchTo64BitMode. The only real downside I see to X86SwitchTo64BitMode is that it's relative to the thread's stack. Where as KiFastSystemCall, all the threads rely on a single global function.

excr4sh - July 1, 2015, 1:37 am

(Jul 1, 2015 01:02 AM)RISC Wrote: [\[CITE\]](#) You can make a two byte relative jump to KiIntSystemCall and perform the five byte patch from there. It's more straight forward than X86SwitchTo64BitMode. The only real downside I see to X86SwitchTo64BitMode is that it's relative to the thread's stack. Where as KiFastSystemCall, all the threads rely on a single global function. You talk about this:

```
0:000:x86> dt ntdll32!_TEB
+0x000 NtTib      : _NT_TIB
+0x01c EnvironmentPointer : Ptr32 Void
+0x020 ClientId   : _CLIENT_ID
+0x028 ActiveRpcHandle : Ptr32 Void
+0x02c ThreadLocalStoragePointer : Ptr32 Void
+0x030 ProcessEnvironmentBlock : Ptr32 _PEB
+0x034 LastErrorValue : Uint4B
+0x038 CountOfOwnedCriticalSections : Uint4B
+0x03c CsrClientThread : Ptr32 Void
+0x040 Win32ThreadInfo : Ptr32 Void
+0x044 User32Reserved : [26] Uint4B
+0x0ac UserReserved   : [5] Uint4B
+0x0c0 WOW32Reserved  : Ptr32 Void
```

The real Hook is set at wow64cpu!X86SwitchTo64BitMode, WOW32Reserved is a pointer which points to that direction.

RISC - July 1, 2015, 1:49 am

(Jul 1, 2015 01:37 AM)FordCI Wrote: [\[CITE\]](#) You talk about this:

```
0:000:x86> dt ntdll32!_TEB
+0x000 NtTib      : _NT_TIB
+0x01c EnvironmentPointer : Ptr32 Void
+0x020 ClientId   : _CLIENT_ID
+0x028 ActiveRpcHandle : Ptr32 Void
+0x02c ThreadLocalStoragePointer : Ptr32 Void
+0x030 ProcessEnvironmentBlock : Ptr32 _PEB
+0x034 LastErrorValue : Uint4B
+0x038 CountOfOwnedCriticalSections : Uint4B
+0x03c CsrClientThread : Ptr32 Void
+0x040 Win32ThreadInfo : Ptr32 Void
+0x044 User32Reserved : [26] Uint4B
+0x0ac UserReserved   : [5] Uint4B
+0x0c0 WOW32Reserved  : Ptr32 Void
```

The real Hook is set at wow64cpu!X86SwitchTo64BitMode, WOW32Reserved is a pointer which points to that direction.

Ah, I wasn't reading the code correctly. I thought you were overwriting the pointer itself.

excr4sh - July 1, 2015, 2:18 am

(Jul 1, 2015 01:49 AM)RISC Wrote: [\[CITE\]](#) Ah, I wasn't reading the code correctly. I thought you were overwriting the pointer itself. Imagine that one thread calls a function that goes through the hook, in my proxy or sub-sequent calls you can make a call to some function of the system such as CreateFile, that call would pass right through NtCreateFile -> FS: [0xC0] -> My hook, proxy function -> and back to CreateFile, This would cause a infinite loop and in most cases, a Stack Overflow that is why I modify the pointer temporarily.

Only3 - July 1, 2015, 2:42 am

good share if RISC didnt kill it must be hq

Mouse? - July 1, 2015, 6:38 am

Oh, lookie, one of those rare actually security related posts. I don't know anything about injection past the 5 byte jmp so I'll just crawl back into my mousehole.

RISC - July 1, 2015, 8:59 am

(Jul 1, 2015 02:18 AM)FordCI Wrote: [\[CITE\]](#) Imagine that one thread calls a function that goes through the hook, in my proxy or sub-sequent calls you can make a call to some function of the system such as CreateFile, that call would pass right through NtCreateFile -> FS: [0xC0] -> My hook, proxy function -> and back to CreateFile, This would cause a infinite loop and in most cases, a Stack Overflow that is why I modify the pointer temporarily.

You could easily avoid the infinite loop.

C0h is a pointer to the function.

Original: Nt->C0h->Switch

Yours:Nt->C0h->Hook

Another option:Nt->My Pointer to C0h->C0h->Switch

Where My Pointer simply handles the stack before it reaches X86SwitchTo64BitMode.

The first scenerio which you're doing is the more practical solution since you'd avoid having to replace the pointer in every thread's context. The second scenerio works just as well, where it's not a complete hook, you'd have to return tot he original x86switchto64bitmode once you've dealt with the argument.

excr4sh - July 1, 2015, 9:32 am

(Jul 1, 2015 08:59 AM)RISC Wrote: [\[CITE\]](#) You could easily avoid the infinite loop.

C0h is a pointer to the function.

Original: Nt->C0h->Switch

Yours:Nt->C0h->Hook

Another option:Nt->My Pointer to C0h->C0h->Switch

Where My Pointer simply handles the stack before it reaches X86SwitchTo64BitMode.

The first scenerio which you're doing is the more practical solution since you'd avoid having to replace the pointer in every thread's context. The second scenerio works just as well, where it's not a complete hook, you'd have to return tot he original x86switchto64bitmode once you've dealt with the argument.By "Nt->My Pointer to C0h->C0h->Switch" you mean to set a hook in the native function?

In any case I am sure there are several workarounds for this issue.

BTW, for those who want to download the project I'm currently having problems with my github account:

[img]<http://s29.postimg.org/pm3jixsbb/image.jpg>[/img]

Project:

<https://www.sendspace.com/file/ga7lm8>

Mouse? - July 1, 2015, 9:51 am

I'd take that as a compliment that you're writing code that only robots could write.

hezboula123 - July 1, 2015, 11:24 am

Code mov dword ptr fs:[0xC0], edi

push eax

call IndexLookupPrint

mov dword ptr fs:[0xC0], esi

So let's get this straight, if you're going to use any functions such as createfile as you suggested earlier to prevent a catastrophic infinite loop you'd just temporarily switch the pointer then you'd call createfile under the mov dword ptr fs:[0xc0], edi

Seems like a good solution but what happens if let's say another thread calls any function while your pointer is replaced, you won't get the call.

while 0CH = your actual switch trampoline, it won't touch the hook.

excr4sh - July 1, 2015, 11:33 am

(Jul 1, 2015 11:24 AM)hezboula123 Wrote: [\[CITE\]](#) Seems like a good solution but what happens if let's say another thread calls any function while your pointer is replaced, you won't get the call. Each thread has its own TEB:

0:012> ~

```
0 Id: 668.20c Suspend: 1 Teb: 000000`7efdb000 Unfrozen
1 Id: 668.ea4 Suspend: 1 Teb: 000000`7efd8000 Unfrozen
2 Id: 668.df0 Suspend: 1 Teb: 000000`7efd5000 Unfrozen
3 Id: 668.f78 Suspend: 1 Teb: 000000`7efad000 Unfrozen
4 Id: 668.ac Suspend: 1 Teb: 000000`7efaa000 Unfrozen
5 Id: 668.69c Suspend: 1 Teb: 000000`7efa7000 Unfrozen
6 Id: 668.dbc Suspend: 1 Teb: 000000`7efa4000 Unfrozen
7 Id: 668.c94 Suspend: 1 Teb: 000000`7efa1000 Unfrozen
8 Id: 668.cc4 Suspend: 1 Teb: 000000`7ef9e000 Unfrozen
9 Id: 668.cec Suspend: 1 Teb: 000000`7ef9b000 Unfrozen
10 Id: 668.4b0 Suspend: 1 Teb: 000000`7ef98000 Unfrozen
11 Id: 668.8b8 Suspend: 1 Teb: 000000`7ef95000 Unfrozen
. 12 Id: 668.f9c Suspend: 1 Teb: 000000`7ef92000 Unfrozen
```

On WoW64 Processes, threads have two TEB:

0:012> !wow64exts.info

PEB32: 0x7efde000

PEB64: 0x7efdf000

Wow64 information for current thread:

TEB32: 0x7ef94000

TEB64: 0x7ef92000

32 bit, StackBase : 0x5110000

StackLimit : 0x510c000

Deallocation: 0x5010000

64 bit, StackBase : 0x464fd20
StackLimit : 0x464c000
Deallocation: 0x4610000

hezboula123 - July 1, 2015, 11:36 am

(Jul 1, 2015 11:33 AM)FordCl Wrote: [\[CITE\]](#) Each thread has its own TEB:

0:012> ~

```
0 Id: 668.20c Suspend: 1 Teb: 000000`7efdb000 Unfrozen
1 Id: 668.ea4 Suspend: 1 Teb: 000000`7efd8000 Unfrozen
2 Id: 668.df0 Suspend: 1 Teb: 000000`7efd5000 Unfrozen
3 Id: 668.f78 Suspend: 1 Teb: 000000`7efad000 Unfrozen
4 Id: 668.ac Suspend: 1 Teb: 000000`7efaa000 Unfrozen
5 Id: 668.69c Suspend: 1 Teb: 000000`7efa7000 Unfrozen
6 Id: 668.dbc Suspend: 1 Teb: 000000`7efa4000 Unfrozen
7 Id: 668.c94 Suspend: 1 Teb: 000000`7efa1000 Unfrozen
8 Id: 668.cc4 Suspend: 1 Teb: 000000`7ef9e000 Unfrozen
9 Id: 668.cec Suspend: 1 Teb: 000000`7ef9b000 Unfrozen
10 Id: 668.4b0 Suspend: 1 Teb: 000000`7ef98000 Unfrozen
11 Id: 668.8b8 Suspend: 1 Teb: 000000`7ef95000 Unfrozen
. 12 Id: 668.f9c Suspend: 1 Teb: 000000`7ef92000 Unfrozen
```

So from what I understand the TEB contains a pointer to the switch function,

So initially you hooked the switch function (pointer obtained from the current thread's TEB).

I understand. Because RISC was suggesting to replace this pointer in every TEB, but instead you hooked the function.

So once you replace the pointer, it's only replaced for that threads TEB. Then you restore it afterwards while every other thread still has the pointer to switch in their TEB.

Code FS:[0xC0] 4 NT Reserved for Wow64. Contains a pointer to FastSysCall in Wow64.

Edit:

PEB64: 0x7efdf000

Yeah, the x64 TEB would be used for segment switches from wow64 to obtain the pointer to PEB, and find ntdll's x64 address. Then you'd just create a generic call for _stdcall functions and handle the stack based on an 8 byte alignment (x64)

excr4sh - July 1, 2015, 11:52 am

(Jul 1, 2015 11:36 AM)hezboula123 Wrote: [\[CITE\]](#) Because RISC was suggesting to replace this pointer in every TEB, but instead you hooked the function. Yeah, it has some advantages and some disadvantages, it may be able to avoid some anti-hook tools but it will only work per thread so I think it depends on your needs.

MalwareTech - July 1, 2015, 2:05 pm

NTSTATUS safe_NtCreateFile(...)

```
{
mov eax, [NtCreateFile_id]
xor ecx, ecx
lea edx, [esp+4]
call X86SwitchTo64BitMode
add esp, 4
ret 0x2C
}
```

Problem solved.

Only3 - July 1, 2015, 2:24 pm

(Jul 1, 2015 02:05 PM)# Wrote: [\[CITE\]](#) NTSTATUS safe_NtCreateFile(...)
{
mov eax, [NtCreateFile_id]
xor ecx, ecx
lea edx, [esp+4]
call X86SwitchTo64BitMode
add esp, 4
ret 0x2C
}

Problem solved.

? hezboula123 Offline >> closed
he will come back to reply ?

excr4sh - July 1, 2015, 2:30 pm

(Jul 1, 2015 02:05 PM)# Wrote: [\[CITE\]](#) NTSTATUS safe_NtCreateFile(...)
{
mov eax, [NtCreateFile_id]
xor ecx, ecx
lea edx, [esp+4]
call X86SwitchTo64BitMode
add esp, 4
ret 0x2C
}

Problem solved.yeah that will work for the NtCreateFile function, but what if I call some other function that internally call NtCreateFile?

MalwareTech - July 1, 2015, 2:34 pm

(Jul 1, 2015 02:30 PM)FordCI Wrote: [\[CITE\]](#) yeah that will work for the NtCreateFile function, but what if I call some other function that internally call NtCreateFile?
You don't call non native functions within a native hook procedure.

excr4sh - July 1, 2015, 3:04 pm

(Jul 1, 2015 02:34 PM)# Wrote: [\[CITE\]](#) You don't call non native functions within a native hook procedure.Yeah, that is ok or maybe not, one can not always use the native API for example -> CreateProcess, you could implement that function or try it, but creating process involves many issues that are not documented eg initial values of the thread, CSRSS API numbers, CSRSS structures, etc. all that can be implemented by reverse engineering but I do not recommend it, will make my application unsafe, not portable.

Anyway in my project, I use several Win32 functions that might internally call the native API, that's why I decided to modify the value in the TEB so it depends on what type of project you are coding (if it is malware or only as a proof of concept).

skid i dont know assembly - July 1, 2015, 3:05 pm

FordCL. When dumping the arguments from Edx, how do you know the amount of arguments for each ordinal?

MalwareTech - July 1, 2015, 3:17 pm

(Jul 1, 2015 03:04 PM)FordCI Wrote: [\[CITE\]](#) Yeah, that is ok or maybe not, one can not always use the native API for example -> CreateProcess, you could implement that function or try it, but creating process involves many issues that are not documented eg initial values of the thread, CSRSS API numbers, CSRSS structures, etc. all that can be implemented by reverse engineering but I do not recommend it, will make my application unsafe, not portable.

Anyway in my project, I use several Win32 functions that might internally call the native API, that's why I decided to modify the value in the TEB so it depends on what type of project you are coding (if it is malware or only as a proof of concept).

Why would you be calling CreateProcess from within a hook procedure?

excr4sh - July 1, 2015, 3:33 pm

(Jul 1, 2015 03:05 PM)skid i dont know assembly Wrote: [\[CITE\]](#) FordCL. When dumping the arguments from Edx, how do you know the amount of arguments for each ordinal?There is no function, but you can check ret/retn instruction when analyzing.

(Jul 1, 2015 03:17 PM)# Wrote: [\[CITE\]](#) Why would you be calling CreateProcess from within a hook procedure?Take that function as an example there are many functions that call native API, not only Kernel32 but Advapi32, User32, Ws2_32 ...

skid i dont know assembly - July 1, 2015, 3:34 pm

(Jul 1, 2015 03:33 PM)FordCI Wrote: [\[CITE\]](#) There is no function, but you can check ret/retn instruction when analyzing.

Take that function as an example there are many functions that call native API, not only Kernel32 but Advapi32, User32, Ws2_32 ...

You're right, thats exactly what i was thinking. Disassemble the ret/retn instruction and divide by 4 since its stdcall.

(Jul 1, 2015 03:17 PM)# Wrote: [\[CITE\]](#) Why would you be calling CreateProcess from within a hook procedure?

he used that as an example. Regardless his solution works fine

MalwareTech - July 1, 2015, 3:37 pm

(Jul 1, 2015 03:33 PM)FordCI Wrote: [\[CITE\]](#) There is no function, but you can check ret/retn instruction when analyzing.

Take that function as an example there are many functions that call native API, not only Kernel32 but Advapi32, User32, Ws2_32 ...

There is no need to call non native functions from a native hook procedure, not to mention it's unsafe.

Microwave89 - July 2, 2015, 1:32 am

FordCL: Tell me a situation in which you need to hook NTAPI of 32 bit processes on x86-64 machines?

If for concealment reasons you'd better forget about it..

Best Regards - Microwave89

excr4sh - July 2, 2015, 11:42 am

(Jul 2, 2015 01:32 AM)Microwave89 Wrote: [\[CITE\]](#) FordCL: Tell me a situation in which you need to hook NTAPI of 32 bit processes on x86-64 machines?

If for concealment reasons you'd better forget about it..

Best Regards - Microwave89Whether or not there, that's not the point of my thread, I created this project as a proof of concept to prove that this is possible.

G1gg1L3s - November 10, 2015, 10:47 am

(Jul 1, 2015 03:04 PM)FordCI Wrote: [\[CITE\]](#) Yeah, that is ok or maybe not, one can not always use the native API for example -> CreateProcess, you could implement that function or try it, but creating process involves many issues that are not documented eg initial values of the thread, CSRSS API numbers, CSRSS structures, etc. all that can be implemented by reverse engineering but I do not recommend it, will make my application unsafe, not portable.

Anyway in my project, I use several Win32 functions that might internally call the native API, that's why I decided to modify the value in the TEB so it depends on what type of project you are coding (if it is malware or only as a proof of concept).

The problem it's unsafe that's what it is .

//

maltech answered :/

0xDEAD10CC - November 10, 2015, 7:38 pm

(Nov 10, 2015 11:47 AM)smrt Wrote: [\[CITE\]](#) The problem it's unsafe that's what it is .

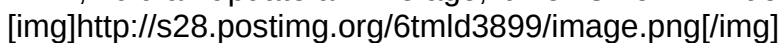
//

maltech answered :/

Thanks for bumping an old thread and pretending that you edited your post even though it shows no edit was made. ?

excr4sh - November 10, 2015, 9:06 pm

(Nov 10, 2015 11:47 AM)smrt Wrote: [\[CITE\]](#) The problem it's unsafe that's what it is .in what sense? just saying that it's unsafe makes no sense.

BTW, I did an update a while ago, it works from Windows XP to Windows 8.1 32/64-bit:


0xDEAD10CC - November 11, 2015, 6:50 pm

^ how come the type is blank on the rest of those? If it works from XP, you should be able to list everything from 16-bit to 64-bit.

excr4sh - November 11, 2015, 9:24 pm

(Nov 11, 2015 07:50 PM)0xDEAD10CC Wrote: [\[CITE\]](#) ^ how come the type is blank on the rest of those? If it works from XP, you should be able to list everything from 16-bit to 64-bit.I call IsWow64Process function to determinate the type of the process (32 or 64-bit), IsWow64Process requires a handle to the process with PROCESS_QUERY_(LIMITED)_INFORMATION access, but you cannot open a handle with that access right to a process with higher integrity (UAC enabled) than the process is running, that is why it is unable to determinate the type of the process. It's not an error/problem, If I run it with admin rights, it shows the type. I took that screenshot from Windows 7 64-bit.

BTW, 16-bit of what?

swag1337 - November 11, 2015, 11:08 pm

Just hook LSTAR and CSTAR

He B TeMy - November 12, 2015, 8:02 am

(Nov 11, 2015 10:24 PM)FordCI Wrote: [\[CITE\]](#) I call IsWow64Process function to determinate the

type of the process (32 or 64-bit), IsWow64Process requires a handle to the process with PROCESS_QUERY_(LIMITED)_INFORMATION access, but you cannot open a handle with that access right to a process with higher integrity (UAC enabled) than the process is running, that is why it is unable to determinate the type of the process. It's not an error/problem, If I run it with admin rights, it shows the type. I took that screenshot from Windows 7 64-bit.

BTW, 16-bit of what?

He meant 16-bit applications, but you can't run them on x64 systems.

excr4sh - November 12, 2015, 7:04 pm

(Nov 12, 2015 09:02 AM)He B TeMy Wrote: [\[CITE\]](#) He meant 16-bit applications, but you can't run them on x64 systems.

Even if I can run them with an emulator, it will not show "16-bit" because Windows doesn't create any process for 16-bit applications, on x86 systems they are emulated by NTVDM.

0xDEAD10CC - November 12, 2015, 7:21 pm

(Nov 12, 2015 08:04 PM)FordCI Wrote: [\[CITE\]](#) Even if I can run them with an emulator, it will not show "16-bit" because Windows doesn't create any process for 16-bit applications, on x86 systems they are emulated by NTVDM.

And your point? How does this prevent you from detecting 16-bit applications? Taskmanager can do it: <https://support.microsoft.com/en-us/kb/320127>

excr4sh - November 12, 2015, 7:34 pm

(Nov 12, 2015 08:21 PM)0xDEAD10CC Wrote: [\[CITE\]](#) And your point? How does this prevent you from detecting 16-bit applications? Taskmanager can do it:

<https://support.microsoft.com/en-us/kb/320127> And yours? What is the relevance of detecting 16-bit applications on my application? Here it is how to list them:

[https://msdn.microsoft.com/en-us/library/bb963830\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/bb963830(v=vs.85).aspx)

easy, right? I could add it to my code, but again, how could it be relevant on this application?

0xDEAD10CC - November 14, 2015, 5:07 pm

(Nov 12, 2015 08:34 PM)FordCI Wrote: [\[CITE\]](#) And yours? What is the relevance of detecting 16-bit applications on my application? Here it is how to list them:

[https://msdn.microsoft.com/en-us/library/bb963830\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/bb963830(v=vs.85).aspx)

easy, right? I could add it to my code, but again, how could it be relevant on this application?

The relevance? Maybe XP perhaps? Since you mentioned it supports XP? Is that not clear?

excr4sh - November 14, 2015, 6:05 pm

(Nov 14, 2015 06:07 PM)0xDEAD10CC Wrote: [\[CITE\]](#) The relevance? Maybe XP perhaps? Since you mentioned it supports XP? Is that not clear?perhaps? are you sure what you are talking about? You should go to OP and reading what this thread is for. this application is not intended to work with 16-bit applications, so why should I list 16-bit applications? and no, the fact that this does not list 16-bit applications does not mean it will not work for Windows XP.

how can you even say that if an application does not list 16-bit applications then will not work on Windows XP?

0xDEAD10CC - November 15, 2015, 10:39 am

(Nov 14, 2015 07:05 PM)FordCI Wrote: [\[CITE\]](#) perhaps? are you sure what you are talking about? You should go to OP and reading what this thread is for. this application is not intended to work with 16-bit applications, so why should I list 16-bit applications? and no, the fact that this does not list 16-bit applications does not mean it will not work for Windows XP.

how can you even say that if an application does not list 16-bit applications then will not work on Windows XP?

Did I say it won't work for XP? Show me where that was mentioned exactly. Additionally, what's the point of listing whether it's 32 or 64 bit if that's not relevant to the program? Otherwise, it clearly is used for some kind of information, so why should 16 be excluded? I never said it was broken, I said that it was missing some functionality that could be added.

excr4sh - November 15, 2015, 12:32 pm

(Nov 15, 2015 11:39 AM)0xDEAD10CC Wrote: [\[CITE\]](#) Did I say it won't work for XP? Show me where that was mentioned exactly. Additionally, what's the point of listing whether it's 32 or 64 bit if that's not relevant to the program? Otherwise, it clearly is used for some kind of information, so why should 16 be excluded? I never said it was broken, I said that it was missing some functionality that could be added. You clearly do not even know what you're talking about. I bet you do not even know how it works.

0xDEAD10CC - November 15, 2015, 1:35 pm

(Nov 15, 2015 01:32 PM)FordCI Wrote: [\[CITE\]](#) You clearly do not even know what you're talking about. I bet you do not even know how it works.

I know a lot more than you do I'd bet, you are the one that doesn't know what you're talking about or you lack the comprehension skills to even read my posts properly. You act like this is advanced for a reason I do not know...

RISC - November 15, 2015, 1:42 pm

(Nov 15, 2015 02:35 PM)0xDEAD10CC Wrote: [\[CITE\]](#) I know a lot more than you do, you are the one that doesn't know what you're talking about or you lack the comprehension skills to even read my posts properly. You act like this is advanced lol.

He did say intercepting system service calls on WoW64 processes. This will not work on 64 bit architectures running 64 bit processes, or 32 bit architectures running 32 or 16 bit processes. The only reason the hook works is because the emulation code has to exit eventually and that's where the hook takes place. If it were to be implemented to work on 16, the concept would be the same, just the code would be slightly different; however, WoW64 process has nothing to do with a 32 bit OS or a 16 bit process on a 32 bit OS.

excr4sh - November 15, 2015, 1:46 pm

(Nov 15, 2015 02:35 PM)0xDEAD10CC Wrote: [\[CITE\]](#) I know a lot more than you do, you are the one that doesn't know what you're talking about or you lack the comprehension skills to even read my posts properly. You act like this is advanced lol. Yeah, you know so much you can even hook X86SwitchTo64BitMode or KiFastSystemCall on native 16-bit applications right? that why you're asking why I don't list 16-bit applications, which I already told you why I don't but you still don't get LOL.

This is not advaced, it's very simple and you don't understand it, but tell us why it would be useful to list 16-bit applications in this code.

0xDEAD10CC - November 15, 2015, 1:48 pm

(Nov 15, 2015 02:42 PM)RISC Wrote: [\[CITE\]](#) He did say intercepting system service calls on WoW64 processes. This will not work on 64 bit architectures running 64 bit processes, or 32 bit architectures running 32 or 16 bit processes. The only reason the hook works is because the emulation code has to exit eventually and that's where the hook takes place. If it were to be implemented to work on 16, the concept would be the same, just the code would be slightly different; however, WoW64 process has nothing to do with a 32 bit OS or a 16 bit process on a 32 bit OS.

He did write this too though:

Quote:BTW, I did an update a while ago, it works from Windows XP to Windows 8.1 32/64-bit:

RISC - November 15, 2015, 1:49 pm

(Nov 15, 2015 02:48 PM)0xDEAD10CC Wrote: [\[CITE\]](#) He did write this too though:
But there are 64 bit Windows XP versions. Which also implement X86SwitchTo64Bit. The hook would work just fine on XP WoW64 processes.

excr4sh - November 15, 2015, 2:22 pm

(Nov 15, 2015 02:48 PM)0xDEAD10CC Wrote: [\[CITE\]](#) He did write this too though: RISC already explained it, additionally, If Windows doesn't use X86SwitchTo64BitMode or KiFastSystemCall then this method will not work and you'll need to hook the functions individually, which is the case with Windows 8 32 bit.

edited:

@0xDEAD10CC, If you still don't understand, # wrote an article about KiFastSystemCall, he explained it very well:: <http://www.#.com/2015/04/intercepting-all-system-calls-by.html>