

Reply to Mr. Gotaimbara,

### IS xAurora Web Browser running in Windows Kernel Mode?

xAurora's core based on Kernel Mode Drivers KMD(s), There are 4 versions of xAurora has been developed by me. xAurora LITE (Published), xAurora RAVEN (Not Published), xAurora ROOKIE (Not Published) and xAurora NIGHTMARE (Not Published). Altogether there are 19 KMD(s) is sitting inside the xAurora to achieve various tasks internally. All internal low level features including EXTREME FEATURES in the browser are control by these KMD(s).

In the Kalinga's blog Mr. Anonymous Skywalker, Mr. Harshadeva Ariyasinghe and Mr. Kalinga Athulathmudali said xAurora LITE is not running in Windows Kernel Mode. I think they don't have much caliber to present the xAurora's KMD(s) PoC to the public community. With the lamer tools like KrView - Kernrate or Kernel Debugger on WinDbg will not work on xAurora. xAurora Browser has its own ANTI DEBUGGER protection scheme applied in the development stage and at the final stage of development I have set the various ANTI DEBUGGING techniques. Therefore above 2 DEBUGGERS can never break the ANTI DEBUGGING Castle Security Scheme of xAurora Web Browser.

In the Kalinga's blog, there is a tutorial "Is xAurora running in Kernel Mode?" written by person called Mr. Anonymous Skywalker trying to prove that the xAurora Web Browser is not running on the Kernel Mode. The person, who's used to monitor and debug the xAurora Web Browser's main executable file, shows the inability and skill of the particular person. These mentioned lamer tools are mostly suitable for CRASHDUMP Analysis and Kernel Process/Hooks Analysis, not suitable for software debugging.

Recommended products to debug the software are IDA Pro 5.x Advanced Extreme Edition, OllyDebug 1.x Debuggers Moded Edition, and Zeta Debugger. These mentioned Debuggers are the best debuggers for analyzing the software. But, unfortunately those mentioned Debuggers are not sufficient to debug/break the xAurora's Main Executable. Because xAurora's has its own ANTI DEBUG Castle Security Protection Scheme and AI World's first Anti-Crack Loader's Code that protecting and wrapping the CODE section of the Main Executable. Only way to Debug/Defeat the xAurora Executable is to run the xAurora in BOOTTIME DEUBBER like SYSER DEBUGGER (Only DEBUGGER in the world that running with HAL and KERNEL real-time, because this DEBUGGER running below the KERNEL (in-between HAL & KERNEL)).

But using this Syser Debugger, you many never can see some internals of the file. Because, most of the internal segments in the main executable file has been encrypted with my own version of encryption algorithm (Datagalaxy Encryption – First AI based Encryption in the World). Anyway try to defeat my protection, if you can. Good luck.

Currently, I have applied following protection schemes to xAurora Main Executable file to achieve maximum protection (Castle Security).

---

Why xAurora's KMD(s) is not visible in the KERNEL MODE DEBUGGERS (Ex. Olly, WinDBG, Datarescue IDA Pro or Softlce)?

I have created an Invisible Kernel Threads and hide all the xAurora KMD(s) Functions inside the INISIBLE HOOK/THREAD Pool in the KERNEL.

### Invisible Kernel Hooks

---

Because a ring3 process has nearly no rights we transfer the action to a KMD in Ring 0.

At first we hook NtQuerySystemInformation. Does a thread call it and specifies the query class 5 (SystemProcessInformation) we modify the returned chain of process information structures. In fact I don't overwrite the whole structure. I enlarge the SYSTEM\_PROCESS\_INFORMATION.SizeOfBlock structure item of the process block being before the block of our process.

This hook isn't realized by a redirection of the API EntryPoint but by modifying the ServiceDescriptorTable whose address one can get from a structure which is addressed by the NtOsKrn!KeServiceDescriptorTable export:

SSDT STRUCT

```
pSSAT      LPVOID ?      ; System Service Address Table ( LPVOID[] )
Obsolete   DWORD  ?      ; or maybe: API ID base
dwAPICount  DWORD  ?
pSSPT      LPVOID ?      ; System Service Parameter Table ( BYTE[] )
```

SSDT ENDS



Hooking NtUserBuildHwndList sounds good for our purposes. NtUserBuildHwndList has 7 arguments and its prototype looks something like:

```
NTSTATUS NTAPI
NtUserBuildHwndList(                                     ; Undocumented
    IN ARGUMENT_1,
    IN hParentHwnd,
    IN BOOL,
    IN ARGUMENT_4,
    IN SpaceForHandlesInBufferCount,
    OUT pOutputBuffer,
    OUT pbResult
);
```

This function isn't exported from win32k.sys. So we need to find its ServiceDescriptorTable. There is an undocumented non-accessible descriptor. The so called ServiceDescriptorTableShadow. I found it some bytes under the address being exported as NtOsKrn!KeServiceDescriptorTable. Little memory snippet...

```
0x0000: SSDT structure for Native API IDs < 0x1000 ; non-shadow SSDT
0x0010: 00000000 00000000 00000000 00000000 ; table terminator
0x0020: 00000000 00000000 00000000 00000000
0x0030: 00000000 00000000 00000000 00000000
0x0040: 00000000 00000000 00000000 00000000
0x0050: SSDT structure for Native API IDs < 0x1000 ; KeServiceDescriptorTableShadow !
0x0060: SSDT structure for Native API IDs >= 0x1000 ; SSDT for win32k.sys
0x0070: 00000000 00000000 00000000 00000000 ; table terminator
```

Now we just need the Native API ID of NtUserBuildHwndList. We've luck.

IDA says:

[...]

```
sub_0_77E0678A proc near
    mov  eax, 112Eh ; EAX == ID of win32k!NtUserBuildHwndList !!!
    lea  edx, [esp+arg_0]
    int  2Eh
    retn 1Ch
sub_0_77E0678A endp
```

```

EnumWindows    proc near
                xor    eax, eax
                push  eax
                push  eax
                push  [esp+8+arg_4]
                push  [esp+0Ch+arg_0]
                push  eax
                push  eax
                call  sub_0_77E06607
                retn  8
EnumWindows    endp

```

[...]

So we can grab the Native API ID of the API directly from above EnumWindows, replace the routine address in the SSDTS.pSSAT[ 0x112E ] and we've hooked the routine successfully.

We can use user32!GetWindowThreadProcessId to decide in KernelMode whether one of the returned window handles belongs to our process or not because this API doesn't call any second API but only the raw NT structures.

## Anti-Debug Techniques

---

(a). PEB.BeingDebugged Flag: IsDebuggerPresent()

This code for identifying if a debugger is present using the IsDebuggerPresent() API and the PEB.BeingDebugged flag:

```

; call kernel32!IsDebuggerPresent()
call    [IsDebuggerPresent]
test    eax,eax
jnz     .debugger_found

; check PEB.BeingDebugged directly
mov     eax,dword [fs:0x30]    ;EAX = TEB.ProcessEnvironmentBlock
movzx   eax,byte [eax+0x02]    ;AL = PEB.BeingDebugged
test    eax,eax
jnz     .debugger_found

```

(b). PEB.NtGlobalFlag, Heap Flags

Checks if PEB.NtGlobalFlag is not equal to 0, and if additional flags are set PEB.ProcessHeap:

```
;ebx = PEB
mov     ebx,[fs:0x30]

;Check if PEB.NtGlobalFlag != 0
cmp     dword [ebx+0x68],0
jne     .debugger_found

;eax = PEB.ProcessHeap
mov     eax,[ebx+0x18]

;Check PEB.ProcessHeap.Flags
cmp     dword [eax+0x0c],2
jne     .debugger_found

;Check PEB.ProcessHeap.ForceFlags
cmp     dword [eax+0x10],0
jne     .debugger_found
```

(c). DebugPort: CheckRemoteDebuggerPresent() / NtQueryInformationProcess()

Typical call to CheckRemoteDebuggerPresent() and NtQueryInformationProcess () to detect if the current process is being debugged:

```
; using kernel32!CheckRemoteDebuggerPresent()
lea     eax,[.bDebuggerPresent]
push   eax                ;pbDebuggerPresent
push   0xffffffff        ;hProcess
call   [CheckRemoteDebuggerPresent]
cmp     dword [.bDebuggerPresent],0
jne     .debugger_found
```

```

; using ntdll!NtQueryInformationProcess(ProcessDebugPort)
lea    eax,[.dwReturnLen]
push   eax                                ;ReturnLength
push   4                                  ;ProcessInformationLength
lea    eax,[.dwDebugPort]
push   eax                                ;ProcessInformation
push   ProcessDebugPort                   ;ProcessInformationClass (7)
push   0xffffffff                          ;ProcessHandle
call   [NtQueryInformationProcess]
cmp    dword [.dwDebugPort],0
jne    .debugger_found

```

#### (d). Debugger Interrupts

Sets the value of EAX to 0xFFFFFFFF (via the CONTEXT6 record) while inside exception handler to signify that the exception handler had been called:

```

;set exception handler
push   .exception_handler
push   dword [fs:0]
mov    [fs:0], esp

;reset flag (EAX) invoke int3
xor    eax,eax
int3

;restore exception handler
pop    dword [fs:0]
add    esp,4

;check if the flag had been set
test   eax,eax
je     .debugger_found

...

```

```

.exception_handler:
;EAX = ContextRecord
mov     eax,[esp+0xc]
;set flag (ContextRecord.EAX)
mov     dword [eax+0xb0],0xffffffff
;set ContextRecord.EIP
inc     dword [eax+0xb8]
xor     eax,eax
retn

```

#### (e). Timing Checks

It uses the RDTSC (Read Time-Stamp Counter) instruction before and after several instructions, and then computes the delta. The delta value of 0x200 depends on how much code is executed between the two RDTSC instructions.

```

rdtsc
mov     ecx,eax
mov     ebx,edx

;... more instructions
nop
push   eax
pop    eax
nop
;... more instructions

;compute delta between RDTSC instructions
rdtsc

;Check high order bits
cmp    edx,ebx
ja     .debugger_found
;Check low order bits
sub    eax,ecx
cmp    eax,0x200
ja     .debugger_found

```

#### (f). SeDebugPrivilege

Some packers indirectly use SeDebugPrivilege to identify if the process is being debugged by attempting to open the CSRSS.EXE process.

```
;query for the PID of CSRSS.EXE
    call     [CsrGetProcessId]

;try to open the CSRSS.EXE process
push     eax
push     FALSE
push     PROCESS_QUERY_INFORMATION
call     [OpenProcess]

;if OpenProcess() was successful,
; Process is probably being debugged
test     eax,eax
jnz     .debugger_found
```

#### (g). DebugObject: NtQueryObject()

The number of DebugObject can be obtained by querying information about all object types using ntdll!NtQueryObject(). NtQueryObject accepts 5 parameters, and for the purpose of querying all objects types, the ObjectHandle parameter is set to NULL and ObjectInformationClass is to ObjectAllTypeInfoInformation (3):

```
NTSTATUS NTAPI NtQueryObject(
    HANDLE   ObjectHandle,
    OBJECT_INFORMATION_CLASS ObjectInformationClass,
    PVOID    ObjectInformation,
    ULONG    Length,
    PULONG   ResultLength
)
```

The said API returns an OBJECT\_ALL\_INFORMATION structure, in which the NumberOfObjectsTypes field is the count of total object types in the ObjectTypeInfoInformation array:

```
typedef struct _OBJECT_ALL_INFORMATION {
    ULONG NumberOfObjectsTypes;
    OBJECT_TYPE_INFORMATION ObjectTypeInformation[1];
}
```

The detection routine will then iterate thru the ObjectTypeInformation array which has the following structure:

```
typedef struct _OBJECT_TYPE_INFORMATION {
    [00] UNICODE_STRING TypeName;
    [08] ULONG TotalNumberOfHandles;
    [0C] ULONG TotalNumberOfObjects;
    ... More fields...
}
```

The TypeName field is then compared to the UNICODE string "DebugObject", and then the TotalNumberOfObjects or the TotalNumberOfHandles field is checked for a non-zero value.

#### (h). Debugger Window

Uses FindWindow() to identify if OllyDbg or WinDbg is running in the system via the windows they create:

```
push    NULL
push    .szWindowClassOllyDbg
call    [FindWindowA]
test    eax,eax
jnz     .debugger_found
```

```
push    NULL
push    .szWindowClassWinDbg
call    [FindWindowA]
test    eax,eax
jnz     .debugger_found
```

```
.szWindowClassOllyDbg    db "OLLYDBG",0
```

```
.szWindowClassWinDbg    db "WinDbgFrameClass",0
```

### (i). Debugger Process

If a debugger is running in the system is to list all process and check if the process name is that of a debugger (e.g. OLLYDBG.EXE, windbg.exe, etc.) The implementation is straight forward and just involves using Process32First/Next() and then Checking if the image name is that of a debugger.

Some packers also go as far as reading a process' memory using kernel32!ReadProcessMemory() and then search for debugger-related strings (e.g. "OLLYDBG") in case the reverser renames the debugger's executable. Once a debugger is found, the packer may display an error message, silently exit or terminate the debugger.

### (j). Device Drivers

Detecting if a kernel mode debugger is active in the system is to try accessing their device drivers. The technique is fairly simple and just involves calling kernel32!CreateFile() against well-known device names used by kernel mode debuggers such as SoftICE.

```
push    NULL
push    0
push    OPEN_EXISTING
push    NULL
push    FILE_SHARE_READ
push    GENERIC_READ
push    .szDeviceNameNtice
call    [CreateFileA]
cmp     eax,INVALID_HANDLE_VALUE
jne     .debugger_found
```

```
.szDeviceNameNtice db "\\.\NTICE",0
```

### (k). OllyDbg: Guard Pages

The code allocates a memory, store code in the allocated memory, and then enable the PAGE\_GUARD attribute. It then initializes its marker (EAX) to 0, and trigger the STATUS\_GUARD\_PAGE\_VIOLATION by executing code in the page guarded allocated memory. If the code is being debugged in OllyDbg, the marker will be unchanged since the exception handler will not be called.

```

; set up exception handler
push  .exception_handler
push  dword [fs:0]
mov   [fs:0], esp

; allocate memory
push  PAGE_READWRITE
push  MEM_COMMIT
push  0x1000
push  NULL
call  [VirtualAlloc]
test  eax,eax

jz    .failed
mov   [.pAllocatedMem],eax

; store a RETN on the allocated memory
mov   byte [eax],0xC3

; then set the PAGE_GUARD attribute of the allocated memory
lea   eax,[.dwOldProtect]
push  eax
push  PAGE_EXECUTE_READ | PAGE_GUARD
push  0x1000
push  dword [.pAllocatedMem]
call  [VirtualProtect]

; set marker (EAX) as 0
xor   eax,eax
; trigger a STATUS_GUARD_PAGE_VIOLATION exception
call  [.pAllocatedMem]
; check if marker had not been changed (exception handler not called)
test  eax,eax
je    .debugger_found
...

```

```

.exception_handler
;EAX = CONTEXT record
mov     eax,[esp+0xc]
;set marker (CONTEXT.EAX) to 0xffffffff
; to signal that the exception handler was called
mov     dword [eax+0xb0],0xffffffff
xor     eax,eax
retn

```

## Anti-Breakpoint & Patching Techniques

---

### (a). Software Breakpoint Detection

breakpoints which are set by modifying the code at the target address, replacing it with a byte value 0xCC (INT3 / Breakpoint Interrupt). Packers identify software breakpoints by scanning for the byte 0xCC in the protector code and/or an API code.

```

cld
mov     edi,Protected_Code_Start
mov     ecx,Protected_Code_End - Protected_Code_Start
mov     al,0xcc
repne  scasb
jz     .breakpoint_found

```

### (b). Hardware Breakpoint Detection

Detecting hardware breakpoints requires a bit of code to perform since debug registers are not accessible in Ring 3. Thus, packers utilize the CONTEXT structure which contains the values of the debug registers. The CONTEXT structure is accessed via the ContextRecord parameter passed to an exception handler.

```

; set up exception handler
push   .exception_handler
push   dword [fs:0]
mov    [fs:0], esp

; eax will be 0xffffffff if hardware breakpoints are identified
xor    eax,eax

```

```
; throw an exception
mov    dword [eax],0

; restore exception handler
pop    dword [fs:0]
add    esp,4

; test if EAX was updated (breakpoint identified)
test   eax,eax
jnz    .breakpoint_found
```

```
:::
```

```
.exception_handler
```

```
;EAX = CONTEXT record
mov    eax,[esp+0xc]

;check if Debug Registers Context.Dr0–Dr3 is not zero
cmp    dword [eax+0x04],0
jne    .hardware_bp_found
cmp    dword [eax+0x08],0
jne    .hardware_bp_found
cmp    dword [eax+0x0c],0
jne    .hardware_bp_found
cmp    dword [eax+0x10],0
jne    .hardware_bp_found
jmp    .exception_ret
```

```
.hardware_bp_found
```

```
;set Context.EAX to signal breakpoint found
mov    dword [eax+0xb0],0xffffffff
```

```
.exception_ret
```

```
;set Context.EIP upon return
add    dword [eax+0xb8],6
xor    eax,eax
retn
```

### (c). Patching Detection via Code Checksum Calculation

Patching detection tries to identify if a part of the packer code had been modified which suggests that anti-debugging routines may have been disabled, and as a second purpose can identify if software breakpoints are set. Patching detection is implemented via code checksum, and the checksum calculation can range from simple to intricate checksum/hash algorithms.

```
mov     esi,Protected_Code_Start
mov     ecx,Protected_Code_End - Protected_Code_Start
xor     eax,eax
.checksum_loop
movzx   ebx,byte [esi]
add     eax,ebx
rol     eax,1
inc     esi
loop    .checksum_loop

cmp     eax,dword [.dwCorrectChecksum]
jne     .patch_found
```

### Anti-Analysis Techniques

---

#### (a). Encryption and Compression

Decryption routines are easily recognizable as loops which perform a fetch, compute, and store data operation. Below is an example of a simple decryption routine that performs several XOR operations on an encrypted DWORD value.

```
0040A07C LODS DWORD PTR DS:[ESI]
0040A07D XOR EAX,EBX
0040A07F SUB EAX,12338CC3
0040A084 ROL EAX,10
0040A087 XOR EAX,799F82D0
0040A08C STOS DWORD PTR ES:[EDI]
0040A08D INC EBX
0040A08E LOOPD SHORT 0040A07C ;decryption loop
```

Decryption routine of a polymorphic packers:

```
00476056 MOV BH,BYTE PTR DS:[EAX]
00476058 INC ESI
00476059 ADD BH,0BD
0047605C XOR BH,CL
0047605E INC ESI
0047605F DEC EDX
00476060 MOV BYTE PTR DS:[EAX],BH
00476062 CLC
00476063 SHL EDI,CL
::: More garbage code
00476079 INC EDX
0047607A DEC EDX
0047607B DEC EAX
0047607C JMP SHORT 0047607E
0047607E DEC ECX
0047607F JNZ 00476056 ;decryption loop
```

```
0040C045 MOV CH,BYTE PTR DS:[EDI]
0040C047 ADD EDX,EBX
0040C049 XOR CH,AL
0040C04B XOR CH,0D9
0040C04E CLC
0040C04F MOV BYTE PTR DS:[EDI],CH
0040C051 XCHG AH,AH
0040C053 BTR EDX,EDX
0040C056 MOVSX EBX,CL
::: More garbage code
0040C067 SAR EDX,CL
0040C06C NOP
0040C06D DEC EDI
0040C06E DEC EAX
0040C06F JMP SHORT 0040C071
0040C071 JNZ 0040C045 ;decryption loop
```

(b). Garbage Code and Code Permutation

To illustrate, below is an example of a debugger detection routine which had been permuted and garbage codes inserted in between the permuted instructions:

```
004018A3 MOV EBX,A104B3FA
004018A8 MOV ECX,A104B412
004018AD PUSH 004018C1
004018B2 RETN
004018B3 SHR EDX,5
004018B6 ADD ESI,EDX
004018B8 JMP SHORT 004018BA
004018BA XOR EDX,EDX
004018BC MOV EAX,DWORD PTR DS:[ESI]
004018BE STC
004018BF JB SHORT 004018DE
004018C1 SUB ECX,EBX
004018C3 MOV EDX,9A01AB1F
004018C8 MOV ESI,DWORD PTR FS:[ECX]
004018CB LEA ECX,DWORD PTR DS:[EDX+FFFF7FF7]
004018D1 MOV EDX,600
004018D6 TEST ECX,2B73
004018DC JMP SHORT 004018B3
004018DE MOV ESI,EAX
004018E0 MOV EAX,A35ABDE4
004018E5 MOV ECX,FAD1203A
004018EA MOV EBX,51AD5EF2
004018EF DIV EBX
004018F1 ADD BX,44A5
004018F6 ADD ESI,EAX
004018F8 MOVZX EDI,BYTE PTR DS:[ESI]
004018FB OR EDI,EDI
004018FD JNZ SHORT 00401906

00401081 MOV EAX,DWORD PTR FS:[18]
00401087 MOV EAX,DWORD PTR DS:[EAX+30]
0040108A MOVZX EAX,BYTE PTR DS:[EAX+2]
0040108E TEST EAX,EAX
00401090 JNZ SHORT 00401099
```

### (c). Anti-Disassembly

Simple PEB.BeingDebugged flag check with some anti-disassembly code added. The highlighted lines are the main instructions, while the remaining are the anti-disassembly codes. It uses the garbage byte 0xff and adds fake conditional jump into the garbage byte for disassemblers to follow:

```
;Anti-disassembly sequence #1
```

```
    push    .jmp_real_01
```

```
    stc
```

```
    jnc     .jmp_fake_01
```

```
    retn
```

```
.jmp_fake_01:
```

```
    db     0xff
```

```
.jmp_real_01:
```

```
    mov     eax,dword [fs:0x18]
```

```
;Anti-disassembly sequence #2
```

```
    push    .jmp_real_02
```

```
    clc
```

```
    jc     .jmp_fake_02
```

```
    retn
```

```
.jmp_fake_02:
```

```
    db     0xff
```

```
.jmp_real_02:
```

```
    mov     eax,dword [eax+0x30]
```

```
    movzx   eax,byte [eax+0x02]
```

```
    test    eax,eax
```

```
    jnz     .debugger_found
```

### (a). Misdirection and Stopping Execution via Exceptions

Performs misdirection by throwing an overflow exception (via INTO) when the overflow flag is set by the ROL instruction after loop iteration. But since an overflow exception is a trap exception, EIP will just point to the JMP instruction. If the reverser is using OllyDbg, and the reverser did not pass the exception to the exception handler (via Shift+F7/F8/F9) and just continually performs a step, the reverser will be tracing an endless loop.

```
; set up exception handler
push  .exception_handler
push  dword [fs:0]
mov   [fs:0], esp

; throw an exception
mov   ecx,1
.loop:
rol   ecx,1
into
jmp   .loop

; restore exception handler
pop   dword [fs:0]
add   esp,4
...

.exception_handler
;EAX = CONTEXT record
mov   eax,[esp+0xc]
;set Context.EIP upon return
add   dword [eax+0xb8],2
xor   eax,eax
retn
```

## (b). Blocking Input

BlockInput() takes 1 boolean parameter fBlockIt. If true, keyboard and mouse events are blocked, if false, keyboard and mouse events are unblocked:

```
; Block input
push    TRUE
call    [BlockInput]

; ...Unpacking code...

; Unblock input
push    FALSE
call    [BlockInput]
```

## (c). ThreadHideFromDebugger

A call to the NtSetInformationThread() would be:

```
push    0                ;InformationLength
push    NULL             ;ThreadInformation
push    ThreadHideFromDebugger ;0x11
push    0xffffffff      ;GetCurrentThread()
call    [NtSetInformationThread]
```

## (d). Disabling Breakpoints

The debug registers are cleared via the CONTEXT record passed to the exception handler:

```
; set up exception handler
push    .exception_handler
push    dword [fs:0]
mov     [fs:0], esp

; throw an exception
xor     eax,eax
mov     dword [eax],0
```

```
; restore exception handler
pop    dword [fs:0]
add    esp,4
...
```

```
.exception_handler
```

```
;EAX = CONTEXT record
```

```
mov    eax,[esp+0xc]
```

```
;Clear Debug Registers: Context.Dr0–Dr3,Dr6,Dr7
```

```
mov    dword [eax+0x04],0
```

```
mov    dword [eax+0x08],0
```

```
mov    dword [eax+0x0c],0
```

```
mov    dword [eax+0x10],0
```

```
mov    dword [eax+0x14],0
```

```
mov    dword [eax+0x18],0
```

```
;set Context.EIP upon return
```

```
add    dword [eax+0xb8],6
```

```
xor    eax,eax
```

```
retn
```

(e). Unhandled Exception Filter

In which an top level exception filter is set using `SetUnhandledExceptionFilter()`, and then an access violation is thrown. If the process is being debugged, the debugger will just receive a second chance exception; otherwise, the exception filter will setup `CONTEXT.EIP` and continue the execution.

```
;set the exception filter
```

```
push    .exception_filter
```

```
call    [SetUnhandledExceptionFilter]
```

```
mov     [.original_filter],eax
```

```
;throw an exception
```

```
xor     eax,eax
```

```
mov     dword [eax],0
```

```
;restore exception filter
  push    dword [.original_filter]
  call    [SetUnhandledExceptionFilter]

  ...
```

.exception\_filter:

```
;EAX = ExceptionInfo.ContextRecord
mov     eax,[esp+4]
mov     eax,[eax+4]

;set return EIP upon return
add     dword [eax+0xb8],6

;return EXCEPTION_CONTINUE_EXECUTION
mov     eax,0xffffffff
retn
```

(f). OllyDbg: OutputDebugString() Format String Bug

Causes OllyDbg to throw an access violation or unexpectedly terminate:

```
  push    .szFormatString
  call    [OutputDebugStringA]
  ...
.szFormatString db "%s%s",0
```

### (a). Process Injection

1. Spawn the host process as a suspended child process. This is done using the `CREATE_SUSPENDED` process creation flag passed to `kernel32!CreateProcess()`. At this point an initialization thread is created and suspended, DLLs are still not loaded since the loader routine (`ntdll!LrdInitializeThunk`) is still not called. The context of the said thread is setup such as the register values contains information such as the PEB address, and entry point of the host process.
2. Using `kernel32!GetThreadContext()`, the context of the child process' initialization thread is retrieved
3. The PEB address of the child process is retrieved via `CONTEXT.EBX`
4. The image base of the child process is retrieved by reading `PEB.ImageBase` (`PEB + 0x8`)
5. The original host image in the child process is then unmapped using `ntdll!NtUnmapViewOfSection()` with the `BaseAddress` parameter pointing to the retrieved image base
6. The unpacking stub will then allocate memory inside the child process using `kernel32!VirtualAllocEx()` with `dwSize` parameter equal to the image size of the unpacked executable.
7. Using `kernel32!WriteProcessMemory()`, the PE header and each of the sections of the unpacked executable is written to the child process.
8. The `PEB.ImageBase` of the child process is then updated to match the image base of the unpacked executable.
9. The context of the child process' initialization thread is then updated via `kernel32!SetThreadContext()` in which `CONTEXT.EAX` is set to the entry point of the unpacked executable.
10. Execution of the child process is resumed via `kernel32!ResumeThread()`

## (b). Debugger Blocker

This prevents a reverser from attaching a debugger to a protected process. This protection is implemented thru the use of debugging functions provided by Windows.

Specifically, the unpacking stub acts a debugger (parent process) where it spawns and debugs/controls the child process which contains the unpacked executable.

Since the protected process is already being debugged, attaching a debugger via `kernel32!DebugActiveProcess()` will fail since the corresponding native API, `ntdll!NtDebugActiveProcess()` will return `STATUS_PORT_ALREADY_SET`. Internally, the failure of `NtDebugActiveProcess()` is due to the `DebugPort` field of the `EPROCESS` kernel structure being already set.

## (c). TLS Callbacks

Execute code before the actual entry point is executed. This is achieved thru the use Thread Local Storage (TLS) callback functions. Packers may perform its debugger detection and decryption routines via these callback functions so that the reverser will not be able to trace these routines.

TLS callbacks can be identified using PE file parsing tools such as `pedump`. With `pedump`, the Data Directory entries will display if a TLS directory exists in the executable:

### Data Directory

EXPORT	rva: 00000000	size: 00000000
IMPORT	rva: 00061000	size: 000000E0
...		
TLS	rva: 000610E0	size: 00000018
...		
IAT	rva: 00000000	size: 00000000
DELAY_IMPORT	rva: 00000000	size: 00000000
COM_DESCRIPTOR	rva: 00000000	size: 00000000
unused	rva: 00000000	size: 00000000

Then, the actual contents TLS directory is displayed. The `AddressOfCallBacks` field points to an array of callback functions and is null-terminated:

TLS directory:

StartAddressOfRawData	:	00000000
EndAddressOfRawData	:	00000000
AddressOfIndex	:	004610F8
AddressOfCallBacks	:	004610FC
SizeOfZeroFill	:	00000000
Characteristics	:	00000000

(d). Stolen Bytes

Basically portions of codes of the protected executable (usually few instructions of the entry point) which are removed by the packer and is copied and executed from an allocated memory. This protects the executable in a way that if the protected process is dumped from memory, instructions that had been stolen are not recovered.

An executable's original entry point:

```
004011CB MOV EAX,DWORD PTR FS:[0]
004011D1 PUSH EBP
004011D2 MOV EBP,ESP
004011D4 PUSH -1
004011D6 PUSH 0047401C
004011DB PUSH 0040109A
004011E0 PUSH EAX
004011E1 MOV DWORD PTR FS:[0],ESP
004011E8 SUB ESP,10
004011EB PUSH EBX
004011EC PUSH ESI
004011ED PUSH EDI
```

Same file with the first two instructions stolen Protector:

```
004011CB POP EBX
004011CC CMP EBX,EBX
004011CE DEC ESP
004011CF POP ES

004011D0 JECXZ SHORT 00401169
004011D2 MOV EBP,ESP
004011D4 PUSH -1
004011D6 PUSH 0047401C
004011DB PUSH 0040109A
004011E0 PUSH EAX
004011E1 MOV DWORD PTR FS:[0],ESP
004011E8 SUB ESP,10
004011EB PUSH EBX
004011EC PUSH ESI
004011ED PUSH EDI
```

#### (e). API Redirection

API redirection is a way to prevent a reverser from easily rebuilding the import table of the protected executable. Typically, the original import table is destroyed and calls to APIs are redirected into routines located into an allocated memory, these routines are then responsible for calling the actual API.

code calls the API kernel32!CopyFileA():

```
00404F05 LEA EDI,DWORD PTR SS:[EBP-20C]
00404F0B PUSH EDI
00404F0C PUSH DWORD PTR SS:[EBP-210]
00404F12 CALL <JMP.&KERNEL32.CopyFileA>
```

The call was to a stub that performs a JMP in which the address is referenced from the import table:

```
004056B8 JMP DWORD PTR DS:[<&KERNEL32.CopyFileA>
```

## Hardcore Anti-Reverse Engineering Techniques Implemented

---

- (a). Code Replace Technology
- (b). Force Packing
- (c). Relocating and Re-virginizing PE
- (d). API Wrapping
- (e). Anti-Patching
- (f). Anti-Dumping
- (g). Entry Point Obfuscation
- (h). Metamorph Security
- (i). Nanomite Protection
- (j). Copymem Protection
- (k). Resource Encryption
- (l). Memory Guard
- (m). VM Ware/Virtual PC/Virtual Box Protection
- (n). Advance AI Entrypoint Crypt Loader Code Mutex

1 = B8[EP]FFE0 | MOV EAX, JMP EAX

2 = B8[EP]FFD0 | MOV EAX, CALL EAX

3 = B8FFFFFF004883F80075FAB8[EP]FFE0 | LOOP, THEN JMP

4 = B8[EP]5059FFE1 | SWAP WITH STACK

5 = B9[EP]83C1108BC183E810FFD0 | ARIFMETIC MANIPULATIONS

- (o). Hide Process/Protect Processes
- (p). Prevent from Global/Kernel Hooks
- (q). Anti-Anti-Debug
- (r). Parent Process Enumeration/Emulation
- (s). Hardware Locking
- (t). Anti Sandbox (Anubis, Sandboxie, Norman, CW and Sunbelt)
- (u). Anti Joebox Protection
- (v). Anti Thread Expert
- (w). Anti Filemon / Regmon and Procmon Protection

## Real-Word Implemented Protection Examples

---

### 1<sup>st</sup> Mode – Hide Debugger – For Ollydebug

---

```
.386
.model flat, stdcall
option casemap :none ; case sensitive

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data
    DbgNotFoundTitle db "Debugger status:",0h
    DbgFoundTitle db "Debugger status:",0h
    DbgNotFoundText db "Debugger not found!",0h
    DbgFoundText db "Debugger found!",0h
    OdbgFindWindow db "OLLYDBG",0h
.code
```

start:

; This trick finds OLLYDBG window class. If class is present – debugger is detected.

```
PUSH 0
PUSH offset OdbgFindWindow
CALL FindWindow

TEST EAX,EAX
JNE @DebuggerDetected

PUSH 40h
PUSH offset DbgNotFoundTitle
PUSH offset DbgNotFoundText
PUSH 0
CALL MessageBox
JMP @exit
```

@DebuggerDetected:

```
PUSH 30h
PUSH offset DbgFoundTitle
PUSH offset DbgFoundText
PUSH 0
CALL MessageBox
```

@exit:

```
PUSH 0
CALL ExitProcess
```

end start

---

## 2<sup>nd</sup> Mode – Hide Debugger – IsDebuggerPresent()

---

.386

```
.model flat, stdcall
option casemap :none ; case sensitive
```

```
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
```

```
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
```

.data

```
DbgNotFoundTitle db "Debugger status:",0h
DbgFoundTitle db "Debugger status:",0h
DbgNotFoundText db "Debugger not found!",0h
DbgFoundText db "Debugger found!",0h
```

.code

start:

```
; This trick can detect all OllyDBG that hook IsDebuggerPresent API
; Then IsDebuggerPresent is called. If it returns value different from 0x90 IsDebuggerPresent is hooked!
```

```
ASSUME FS:NOTHING
MOV EAX,DWORD PTR FS:[30h]
LEA EAX,BYTE PTR DS:[EAX+2h]
```

MOV BYTE PTR[EAX],90h

CALL IsDebuggerPresent

CMP EAX,90h

JNE @DebuggerDetected

PUSH 40h

PUSH offset DbgNotFoundTitle

PUSH offset DbgNotFoundText

PUSH 0

CALL MessageBox

JMP @exit

@DebuggerDetected:

PUSH 30h

PUSH offset DbgFoundTitle

PUSH offset DbgFoundText

PUSH 0

CALL MessageBox

@exit:

PUSH 0

CALL ExitProcess

end start

---

### 3<sup>rd</sup> Mode – Hide Debugger – OpenProcess()

---

.386

.model flat, stdcall

option casemap :none ; case sensitive

include \masm32\include\windows.inc

include \masm32\include\user32.inc

include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib

includelib \masm32\lib\kernel32.lib

```
.data
    DbgNotFoundTitle db "Debugger status:",0h
    DbgFoundTitle db "Debugger status:",0h
    DbgNotFoundText db "Debugger not found!",0h
    DbgFoundText db "Debugger found!",0h
    Krnel32 db "kernel32.dll",0h
    OpnProcess db "OpenProcess",0h
.code
```

start:

```
; This trick can detect HideDebugger plugin because it modifies OpenProcess API
; by inserting JMP FAR (0xEA) instruction.
```

```
PUSH offset Krnel32          ;kernel32.dll
CALL GetModuleHandle
```

```
PUSH offset OpnProcess       ;OpenProcess
PUSH EAX
CALL GetProcAddress
```

```
CMP BYTE PTR[EAX+6],0EAh
JE @DebuggerDetected
```

```
PUSH 40h
PUSH offset DbgNotFoundTitle
PUSH offset DbgNotFoundText
PUSH 0
CALL MessageBox
```

```
JMP @exit
```

@DebuggerDetected:

```
PUSH 30h
PUSH offset DbgFoundTitle
PUSH offset DbgFoundText
PUSH 0
CALL MessageBox
```

@exit:

```
PUSH 0
CALL ExitProcess
```

end start

---

.386

.model flat, stdcall  
option casemap :none ; case sensitive

include \masm32\include\windows.inc  
include \masm32\include\user32.inc  
include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib  
includelib \masm32\lib\kernel32.lib

.data

DbgNotFoundTitle db "Debugger status:",0h  
DbgFoundTitle db "Debugger status:",0h  
DbgNotFoundText db "Debugger not found!",0h  
DbgFoundText db "Debugger found!",0h

.code

start:

LEA EAX,DWORD PTR[IsDebuggerPresent+2h]  
MOV EAX,DWORD PTR[EAX]  
MOV EAX,DWORD PTR[EAX]

CMP BYTE PTR[EAX],64h  
JNE @DebuggerDetected

PUSH 40h  
PUSH offset DbgNotFoundTitle  
PUSH offset DbgNotFoundText  
PUSH 0  
CALL MessageBox  
JMP @exit

@DebuggerDetected:

PUSH 30h  
PUSH offset DbgFoundTitle  
PUSH offset DbgFoundText  
PUSH 0  
CALL MessageBox

@exit:

PUSH 0  
CALL ExitProcess

end start

---

---

.386

```
.model flat, stdcall
option casemap :none ; case sensitive
```

```
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
```

```
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
```

.data

```
DbgNotFoundTitle db "Debugger status:",0h
DbgFoundTitle db "Debugger status:",0h
DbgNotFoundText db "Debugger not found!",0h
DbgFoundText db "Debugger found!",0h
Krnal32 db "kernel32.dll",0h
IsDbgPresent db "IsDebuggerPresent",0h
```

.code

start:

```
    PUSH offset Krnal32          ;kernel32.dll
    CALL GetModuleHandle

    PUSH offset IsDbgPresent     ;IsDebuggerPresent
    PUSH EAX
    CALL GetProcAddress

    CMP BYTE PTR[EAX],64h
    JNE @DebuggerDetected

    PUSH 40h
    PUSH offset DbgNotFoundTitle
    PUSH offset DbgNotFoundText
    PUSH 0
    CALL MessageBox
    JMP @exit
```

```

@DebuggerDetected:
    PUSH 30h
    PUSH offset DbgFoundTitle
    PUSH offset DbgFoundText
    PUSH 0
    CALL MessageBox
@exit:
    PUSH 0
    CALL ExitProcess
end start

```

---

## Hardware/Software SysDebug Control Structure Generic Header

---

```

;enum _SYSDBG_COMMAND
SysDbgQueryModuleInformation    equ 0
SysDbgQueryTraceInformation     equ 1 ;DebugGetTraceInformation
SysDbgSetTracepoint            equ 2 ;SetInternalBreakpoint <38h>
SysDbgSetSpecialCall           equ 3 ;+
SysDbgClearSpecialCalls        equ 4 ;+
SysDbgQuerySpecialCalls        equ 5 ;+
SysDbgBreakPoint               equ 6
SysDbgQueryVersion             equ 7
SysDbgReadVirtual              equ 8 ;+ tested
SysDbgWriteVirtual             equ 9 ;+ tested
SysDbgReadPhysical             equ 10 ;+ tested
SysDbgWritePhysical            equ 11 ;+ tested
SysDbgReadControlSpace         equ 12 ;+
SysDbgWriteControlSpace        equ 13 ;+
SysDbgReadIoSpace              equ 14 ;+
SysDbgWriteIoSpace             equ 15 ;+
SysDbgReadMsr                  equ 16 ;+ tested
SysDbgWriteMsr                 equ 17 ;+ tested
SysDbgReadBusData              equ 18 ;+
SysDbgWriteBusData             equ 19 ;+
SysDbgCheckLowMemory           equ 20
SysDbgEnableKernelDebugger     equ 21 ;invalid info class
SysDbgDisableKernelDebugger    equ 22 ;invalid info class
SysDbgGetAutoKdEnable          equ 23
SysDbgSetAutoKdEnable          equ 24
SysDbgGetPrintBufferSize       equ 25
SysDbgSetPrintBufferSize       equ 26
SysDbgGetKdUmExceptionEnable   equ 27
SysDbgSetKdUmExceptionEnable   equ 28
SysDbgGetTriageDump            equ 29
SysDbgGetKdBlockEnable         equ 30
SysDbgSetKdBlockEnable         equ 31

```

;ControlCode=SysDbgReadMsr/SysDbgWriteMsr

DBGKD\_READ\_WRITE\_MSR struct

Msr ULONG ?

Reserved ULONG ?

DataValueLow ULONG ?

DataValueHigh ULONG ?

DBGKD\_READ\_WRITE\_MSR ends

PDBGKD\_READ\_WRITE\_MSR typedef ptr DBGKD\_READ\_WRITE\_MSR

;ControlCode=SysDbgReadVirtual/SysDbgWriteVirtual

DBGKD\_READ\_WRITE\_VIRTUAL struct

VirtualAddress PVOID ?

Buffer PVOID ?

\_Length ULONG ?

DBGKD\_READ\_WRITE\_VIRTUAL ends

PDBGKD\_READ\_WRITE\_VIRTUAL typedef ptr DBGKD\_READ\_WRITE\_VIRTUAL

;ControlCode=SysDbgReadPhysical/SysDbgWritePhysical

DBGKD\_READ\_WRITE\_PHYSICAL struct

AddressLo DWORD ?

AddressHi DWORD ?

Buffer PVOID ?

\_Length ULONG ?

DBGKD\_READ\_WRITE\_PHYSICAL ends

PDBGKD\_READ\_WRITE\_PHYSICAL typedef ptr DBGKD\_READ\_WRITE\_PHYSICAL

;ControlCode=SysDbgSysReadIoSpace/SysDbgSysWriteIoSpace

DBGKD\_READ\_WRITE\_IO\_SPACE struct

IoAddress DWORD ? ;IN: Aligned to NumBYTES,I/O address

Reserved1 PVOID ? ;Never accessed by the kernel

Buffer PVOID ? ;IN (write) or OUT (read): Ptr to buffer

BytesToRead ULONG ? ;IN: # BYTES to read/write. Only use 1, 2, or 4.

Reserved4 DWORD ? ;Must be 1

Reserved5 DWORD ? ;Must be 0

Reserved6 DWORD ? ;Must be 1

Reserved7 DWORD ? ;Never accessed by the kernel

DBGKD\_READ\_WRITE\_IO\_SPACE ends

PDBGKD\_READ\_WRITE\_IO\_SPACE typedef ptr DBGKD\_READ\_WRITE\_IO\_SPACE

```

;ControlCode=SysDbgSysReadControlSpace/SysDbgSysWriteControlSpace
DBGKD_CONTROL_SPACE_STRUCT struct
ControlAddress      DWORD ?   ;Address of the control space
Reserved2           DWORD ?   ;Must be 0
Buffer              PVOID ?   ;Pointer to buffer
_Length            ULONG ?   ;Length of data
ProcessorNumber     ULONG ?   ;Number of the CPU whose control space is to be read
Reserved6           DWORD ?   ;never used by kernel
DBGKD_CONTROL_SPACE_STRUCT ends

```

```

;ControlCode=SysDbgReadBusData/SysDbgWriteBusData
DBGKD_GET_SET_BUS_DATA struct
BusDataType         ULONG ?
BusNumber           ULONG ?
SlotNumber          ULONG ?
Buffer              PVOID ?
_Offset            ULONG ?
_Length            ULONG ?
DBGKD_GET_SET_BUS_DATA ends
PDBGKD_GET_SET_BUS_DATA typedef ptr DBGKD_GET_SET_BUS_DATA

```

```

DBGKD_MAX_INTERNAL_BREAKPOINTS      equ 20

DBGKD_INTERNAL_BP_FLAG_COUNTONLY    equ 1   ;don't count instructions
DBGKD_INTERNAL_BP_FLAG_INVALID      equ 2   ;disabled BP
DBGKD_INTERNAL_BP_FLAG_SUSPENDED    equ 4   ;temporarily suspended
DBGKD_INTERNAL_BP_FLAG_DYING        equ 8   ;kill on exit

```

```

;ControlCode=SysDbgSetTracepoint
DBGKD_SET_INTERNAL_BREAKPOINT32 struct
BreakpointAddress  ULONG ?
Flags              ULONG ?
DBGKD_SET_INTERNAL_BREAKPOINT32 ends
PDBGKD_SET_INTERNAL_BREAKPOINT32 typedef ptr DBGKD_SET_INTERNAL_BREAKPOINT32

```

```

;ControlCode=SysDbgQueryTraceInformation
DBGKD_GET_INTERNAL_BREAKPOINT32 struct
BreakpointAddress  PVOID ?
Flags              DWORD ?   ;DBGKD_INTERNAL_BP_FLAG_
Calls              DWORD ?
MaxCallsPerPeriod  DWORD ?
MinInstructions    DWORD ?
MaxInstructions    DWORD ?
TotalInstructions  DWORD ?
DBGKD_GET_INTERNAL_BREAKPOINT32 ends
PDBGKD_GET_INTERNAL_BREAKPOINT32 typedef ptr DBGKD_GET_INTERNAL_BREAKPOINT32

```

```
;ControlCode=SysDbgSetSpecialCall
DBGKD_SET_SPECIAL_CALL struct
SpecialCall          DWORD ?
DBGKD_SET_SPECIAL_CALL ends
PDBGKD_SET_SPECIAL_CALL typedef ptr DBGKD_SET_SPECIAL_CALL
```

```
;ControlCode=SysDbgQuerySpecialCalls
DBGKD_QUERY_SPECIAL_CALLS struct
NumberOfSpecialCalls  DWORD ?
;SpecialCalls          DWORD 1 DUP (?)
DBGKD_QUERY_SPECIAL_CALLS ends
PDBGKD_QUERY_SPECIAL_CALLS typedef ptr DBGKD_QUERY_SPECIAL_CALLS
```

```
;enum _BUS_DATA_TYPE {
ConfigurationSpaceUndefined  equ -1
Cmos                          equ 0
EisaConfiguration            equ 1
Pos                           equ 2
CbusConfiguration            equ 3
PCIConfiguration              equ 4
VMEConfiguration             equ 5
NuBusConfiguration           equ 6
PCMCIAConfiguration           equ 7
MPIOConfiguration            equ 8
MPSAConfiguration            equ 9
PNPISAConfiguration           equ 10
SgilInternalConfiguration     equ 11
```

```
;ControlCode=SysDbgReadBusData/SysDbgWriteBusData
DBGKD_READ_WRITE_BUS_DATA struct
_Offset          ULONG ?
Buffer           PVOID ?
_Length          ULONG ?
BusDataType      ULONG ? ;BUS_DATA_TYPE
BusNumber        ULONG ?
SlotNumber       ULONG ?
DBGKD_READ_WRITE_BUS_DATA ends
PDBGKD_READ_WRITE_BUS_DATA typedef ptr DBGKD_READ_WRITE_BUS_DATA
```

---

## 6th Mode – DebugBlocker() – Moded

---

.386

.model flat, stdcall

option casemap:none

```
include    \masm32\include\windows.inc
include    \masm32\include\user32.inc
include    \masm32\include\kernel32.inc
include    \masm32\include\gdi32.inc
include    \masm32\include\masm32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\gdi32.lib
includelib \masm32\lib\masm32.lib
```

```
WndProc    PROTO : DWORD, : DWORD, : DWORD, : DWORD
```

```
;--DebugBlocker//
```

```
DebugThread PROTO
```

.DATA

```
szWinCap   DB        "DebugBlocker", 0
szAbout    DB        "DebugBlocker ", 13, 10
           DB        "An application that debugs itself :)", 13, 10
           DB        "DebugBlock", 13, 10, 13, 10
           DB        " DebugBlock ", 0
```

```
;--DebugBlocker//
```

```
szSpace    DB        " ", 0
szQuotes   DB        """"", 0
```

.DATA?

```
hInstance  DD        ?
hDlg       DD        ?
;--DebugBlocker//
Cline      DD        ?
pId        DD        ?
hdbgThread DD        ?
threadId   DD        ?
cpid1      DB        16 DUP(?)
cpid2      DB        16 DUP(?)
szTemp     DB        255 DUP(?)
szAppLong  DB        255 DUP(?)
szAppShort DB        266 DUP(?)
pif        PROCESS_INFORMATION  <?>
sif        STARTUPINFO         <?>
```

```
dbgr      DEBUG_EVENT    <?>
msg       MSG            <?>
```

```
.CONST
```

```
    ID_ICON      EQU      99
    ID_DIALOG    EQU      100
    ID_ABOUT     EQU      101
```

```
.CODE
```

```
START:
```

```
    invoke GetModuleHandle, 0
    mov    hInstance, eax
    invoke GetCommandLine
    mov    Cline, eax
    invoke GetModuleFileName, hInstance, offset szAppLong, sizeof szAppLong
    invoke GetShortPathName, offset szAppLong, offset szAppShort, sizeof szAppShort
    invoke szCopy, offset szAppLong, offset szTemp
    invoke szRemove, Cline, offset szTemp, offset szAppLong
    invoke szRemove, offset szTemp, offset szTemp, offset szSpace
    invoke szRemove, offset szTemp, offset szTemp, offset szQuotes
    invoke szLen, offset szTemp
    .IF eax!=0
        invoke atol, offset szTemp
        mov    pld, eax
        invoke CreateThread, NULL, 0, addr DebugThread, 0, 0, addr threadId
    .ELSE
        invoke GetCurrentProcessId
        mov    pld, eax
        invoke ltoa, pld, offset cpid1
        invoke szCopy, offset szSpace, offset cpid2
        invoke szCatStr, offset cpid2, offset cpid1
        invoke CreateProcess, addr szAppShort, addr cpid2, NULL, NULL, FALSE, 0, NULL, NULL, addr sif,
addr pif
        invoke DialogBoxParam, hInstance, ID_DIALOG, 0, offset WndProc, 0
    .ENDIF
StartLoop:
    invoke GetMessage, ADDR msg, NULL, 0, 0
    cmp    eax, 0
    je     ExitLoop
    invoke TranslateMessage, addr msg
    invoke DispatchMessage, addr msg
    jmp    StartLoop
```

ExitLoop:

```
invoke CloseHandle, pif.hProcess
invoke CloseHandle, pif.hThread
invoke TerminateProcess, pif.hProcess, 0
invoke ExitProcess, 0
mov     eax, msg.wParam
```

DebugThread PROC

```
LOCAL  MainApp:DWORD
```

```
invoke OpenProcess, PROCESS_ALL_ACCESS, FALSE, pld
mov     MainApp, eax
invoke  DebugActiveProcess, pld
.IF eax==0
    invoke TerminateProcess, MainApp, 0
    invoke ExitProcess, 0
.ENDIF
.WHILE TRUE
    invoke WaitForDebugEvent, addr dbgr, INFINITE
    .IF dbgr.dwDebugEventCode==EXIT_PROCESS_DEBUG_EVENT
        invoke ExitProcess, 0
    .ELSE
        invoke ContinueDebugEvent, dbgr.dwProcessId, dbgr.dwThreadId, DBG_CONTINUE
    .ENDIF
.ENDW
xor     eax, eax
ret
```

DebugThread ENDP

WndProc PROC hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam: DWORD

```
mov     eax, uMsg
cmp     eax, WM_COMMAND
jz      _COMMAND
cmp     eax, WM_INITDIALOG
jz      _INITDIALOG
cmp     eax, WM_CLOSE
jz      _CLOSE
```

\_DEFAULT:

```
xor     eax, eax
ret
```

\_CLOSE:

```
invoke EndDialog, hWnd, 0
invoke PostQuitMessage, 0
xor    eax, eax
ret
```

\_INITDIALOG:

```
invoke LoadIcon, hInstance, ID_ICON
invoke SendMessage, hWnd, WM_SETICON, ICON_SMALL, eax
invoke SetWindowText, hWnd, addr szWinCap
push   hWnd
pop    hDlg
xor    eax, eax
ret
```

\_COMMAND:

```
mov    edx, wParam
shr    edx, 16
.if   wParam==ID_ABOUT
    jmp  _COMMAND_ABOUT
.ELSE
    jmp  _DEFAULT
.ENDIF
xor    eax, eax
ret
```

\_COMMAND\_ABOUT:

```
invoke MessageBox, hWnd, offset szAbout, offset szWinCap, MB_OK or MB_ICONINFORMATION
xor    eax, eax
ret
```

WndProc ENDP

END START

---

## 7<sup>th</sup> Mode – SingleStep – IsDebuggerPresent() – Moded

---

.586

.model flat, stdcall

option casemap :none ; case sensitive

include \masm32\include\windows.inc

include \masm32\include\user32.inc

include \masm32\include\kernel32.inc

include \masm32\include\comdlg32.inc

```
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\comdlg32.lib
```

```
.data
    DbgFoundTitle db "Debugger found:",0h
    DbgFoundText db "Debugger has been found!",0h
    DbgNotFoundTitle db "Debugger not found:",0h
    DbgNotFoundText db "Debugger not found!",0h
.code
```

start:

```
; Finds ring3/ring0 debuggers by executing exception with Trap flag.
; We set the trap flag so it will fire on next instruction, if debugger
; Is not present SEH will fire, and if it is present and user chooses a step over debugging mode will be caught!
```

```
    ASSUME FS:NOTHING
    PUSH offset _SehExit
    PUSH DWORD PTR FS:[0]
    MOV FS:[0],ESP
```

```
;    Set Trap flag!
```

```
    PUSHFD
    XOR DWORD PTR[ESP],154h
    POPFD
```

```
;    If SEH doesn't fire you are caught!
```

```
    PUSH 30h
    PUSH offset DbgFoundTitle
    PUSH offset DbgFoundText
    PUSH 0
    CALL MessageBox
    PUSH 0
    CALL ExitProcess
    RET
```

```
_Exit:
    PUSH 40h
    PUSH offset DbgNotFoundTitle
    PUSH offset DbgNotFoundText
    PUSH 0
    CALL MessageBox
    PUSH 0
    CALL ExitProcess
    RET
```

```
_SehExit:
    POP FS:[0]
    ADD ESP,4
    JMP _Exit
```

```
end start
```

---

## 8<sup>th</sup> Mode – Modular Anti Ring 3 Debug – IsDebuggerPresent() – Moded

---

```
.586
.model flat, stdcall
option casemap :none ; case sensitive

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\comdlg32.inc

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\comdlg32.lib

.data
    DbgFoundTitle db "Debugger found:",0h
    DbgFoundText db "Debugger has been found!",0h
    DbgNotFoundTitle db "Debugger not found:",0h
    DbgNotFoundText db "Debugger not found!",0h
    Tries db 30
    Alloc dd ?

.code
start:

    ASSUME FS:NOTHING
    PUSH offset _SehExit
    PUSH DWORD PTR FS:[0]
    MOV FS:[0],ESP
```

```

; Get NtGlobalFlag

MOV EAX,DWORD PTR FS:[30h]

; Get LDR_MODULE

MOV EAX,DWORD PTR[EAX+12]

; If ring3 debugger is present memory will be allocated and it will contain 0xFEEEEEEE bytes at the end
; of; alloc. This will only happen if ring3 debugger is present!. If there is no debugger SEH will fire and
; take control.

_loop:
INC EAX
CMP DWORD PTR[EAX],0FEEEEEEEh
JNE _loop
DEC [Tries]
JNE _loop

PUSH 30h
PUSH offset DbgFoundTitle
PUSH offset DbgFoundText
PUSH 0
CALL MessageBox
PUSH 0
CALL ExitProcess
RET

_Exit:
PUSH 40h
PUSH offset DbgNotFoundTitle
PUSH offset DbgNotFoundText
PUSH 0
CALL MessageBox
PUSH 0
CALL ExitProcess
RET

_SehExit:
POP FS:[0]
ADD ESP,4
JMP _Exit

```

end start

---

```
.386
.model flat, stdcall
option casemap :none ; case sensitive

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data
DbgNotFoundTitle db "Debugger status:",0h
DbgFoundTitle db "Debugger status:",0h
DbgNotFoundText db "Debugger not found!",0h
DbgFoundText db "Debugger found!",0h
.code

start:

; IsDebuggerPresent is function that returns TRUE if debugger is used to monitor application and FALSE
; if not. This function is available in Windows NT only!

CALL IsDebuggerPresent

CMP EAX,1
JE @DebuggerDetected

PUSH 40h
PUSH offset DbgNotFoundTitle
PUSH offset DbgNotFoundText
PUSH 0
CALL MessageBox

JMP @exit
@DebuggerDetected:

PUSH 30h
PUSH offset DbgFoundTitle
PUSH offset DbgFoundText
PUSH 0
CALL MessageBox
```

@exit:

```
PUSH 0  
CALL ExitProcess
```

end start

---

## 10<sup>th</sup> Mode – Basic Anti Ring 3 Debug – IsDebuggerPresent() – Moded

---

.386

```
.model flat, stdcall  
option casemap :none ; case sensitive
```

```
include \masm32\include\windows.inc  
include \masm32\include\user32.inc  
include \masm32\include\kernel32.inc
```

```
includelib \masm32\lib\user32.lib  
includelib \masm32\lib\kernel32.lib
```

.data

```
DbgNotFoundTitle db "Debugger status:",0h  
DbgFoundTitle db "Debugger status:",0h  
DbgNotFoundText db "Debugger not found!",0h  
DbgFoundText db "Debugger found!",0h
```

.code

start:

```
; This example can detect all ring3 debuggers by accessing PEB!BeingDebuged.  
; You can see this code in kernel32.dll!IsDebuggerPresent function.
```

```
ASSUME FS:NOTHING  
MOV EAX,DWORD PTR FS:[18h]  
MOV EAX,DWORD PTR DS:[EAX+30h]  
MOVZX EAX,BYTE PTR DS:[EAX+2h]
```

```
CMP EAX,1  
JE @DebuggerDetected
```

```
PUSH 40h  
PUSH offset DbgNotFoundTitle  
PUSH offset DbgNotFoundText  
PUSH 0  
CALL MessageBox
```

```
JMP @exit
@DebuggerDetected:
```

```
PUSH 30h
PUSH offset DbgFoundTitle
PUSH offset DbgFoundText
PUSH 0
CALL MessageBox
```

```
@exit:
```

```
PUSH 0
CALL ExitProcess
```

```
end start
```

---

## 11<sup>th</sup> Mode – Basic Anti Ring 3 Debug – IsDebuggerPresent() – Moded – Code Ripping Mode

---

```
.386
```

```
.model flat, stdcall
option casemap :none ; case sensitive
```

```
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
```

```
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
```

```
.data
```

```
DbgNotFoundTitle db "Debugger status:",0h
DbgFoundTitle db "Debugger status:",0h
DbgNotFoundText db "Debugger not found!",0h
DbgFoundText db "Debugger found!",0h
```

```
.code
```

```
start:
```

- ; This example can detect all ring3 debuggers by accessing PEB!BeingDebuged.
- ; You can see this code in kernel32.dll!IsDebuggerPresent function.
- ; This is another way of accessing PEB.

```
ASSUME FS:NOTHING
MOV EAX,DWORD PTR FS:[30h]
MOVZX EAX,BYTE PTR DS:[EAX+2h]
```

```
CMP EAX,1
JE @DebuggerDetected
```

```
PUSH 40h
PUSH offset DbgNotFoundTitle
PUSH offset DbgNotFoundText
PUSH 0
CALL MessageBox
```

```
JMP @exit
```

```
@DebuggerDetected:
```

```
PUSH 30h
PUSH offset DbgFoundTitle
PUSH offset DbgFoundText
PUSH 0
CALL MessageBox
```

```
@exit:
```

```
PUSH 0
CALL ExitProcess
```

```
end start
```

---

## 12<sup>th</sup> Mode – Anti Ring 0 Debug – IsDebuggerPresent() – RemoteDebugger() – Moded

---

```
.386
```

```
.model flat, stdcall
option casemap :none ; case sensitive
```

```
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
```

```
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
```

```
.data
    DbgNotFoundTitle db "Debugger status:",0h
    DbgFoundTitle db "Debugger status:",0h
    DbgNotFoundText db "Debugger not found!",0h
    DbgFoundText db "Debugger found!",0h
    krnl db "kernel32.dll",0h
    chkrdbg db "CheckRemoteDebuggerPresent",0h
```

```
.data?
    IsItPresent dd ?
```

```
.code
```

```
start:

;    CheckRemoteDebuggerPresent is function similar to IsDebuggerPresent. This function is available only
;    in Windows NT and it outputs TRUE or FALSE value if debugger is present in selected process.
;    Load the function via GetProcAddress

    PUSH offset krnl            ;kernel32.dll
    CALL LoadLibrary

    PUSH offset chkrdbg        ;CheckRemoteDebuggerPresent
    PUSH EAX
    CALL GetProcAddress

;    IsItPresent variable will store the result

    PUSH offset IsItPresent
    PUSH -1
    CALL EAX

    MOV EAX,DWORD PTR[IsItPresent]
    TEST EAX,EAX
    JNE @DebuggerDetected

    PUSH 40h
    PUSH offset DbgNotFoundTitle
    PUSH offset DbgNotFoundText
    PUSH 0
    CALL MessageBox

    JMP @exit
```

@DebuggerDetected:

```
PUSH 30h
PUSH offset DbgFoundTitle
PUSH offset DbgFoundText
PUSH 0
CALL MessageBox
```

@exit:

```
PUSH 0
CALL ExitProcess
```

end start

---

## 13<sup>th</sup> Mode – Anti Ring 0 Debug – IsDebuggerPresent() – Generic Anti-Debugging – Olly DLL

.586p

```
.model flat, stdcall           ; 32 bit memory model
option scoped                  ; local labels are enabled, global labels inside
                               ; PROC should be defined with double colons (LABEL::)
option casemap :none          ; case sensitive
```

```
DlgDumpProc      proto :DWORD,:DWORD,:DWORD,:DWORD
```

```
DlgOptionProc    proto :DWORD,:DWORD,:DWORD,:DWORD
```

```
include windows.inc
include kernel32.inc
include user32.inc
include ..\plugin.inc
include masm32.inc
include comdlg32.inc
```

```
includelib kernel32.lib
includelib user32.lib
includelib ..\ollydbg.lib
includelib masm32.lib
includelib comdlg32.lib
```

```
;-----  
; literal string MACRO  
;-----  
literal MACRO quoted_text:VARARG  
    LOCAL    local_text  
    .data  
        local_text db quoted_text,0  
    .code  
    EXITM    <local_text>  
ENDM
```

```
CTEXT    MACRO quoted_text:VARARG  
    EXITM    <offset literal(quoted_text)>  
ENDM
```

```
m2m MACRO    M1, M2  
    push    M2  
    pop     M1  
ENDM
```

```
return MACRO    arg  
    mov     eax, arg  
    ret  
ENDM
```

MAXSIZE equ 260

```
;Dumper.dlg  
IDD_DUMP equ 1000  
IDC_BTNCANCEL equ 1001  
IDC_BTNDUMP equ 1002  
IDC_EDTSIZE equ 1003  
IDC_STC1 equ 1004  
IDC_EDTOFFSET equ 1005  
IDC_STC2 equ 1006
```

```
;Res\Params.dlg  
IDD_OPTION equ 1001  
IDC_GRP1 equ 1003  
IDC_CHK1 equ 1001  
IDC_STC3 equ 1002  
IDC_SLEEPTIME equ 1004  
IDC_STC4 equ 1005  
IDC_BTN1 equ 1006  
IDC_BTN2 equ 1007
```

```
;IsDebug.rc
```

```
.data
the_byte          db 1
null_byte         db 0
ofn               OPENFILENAME <>
FilterString      db "Bin Files",0 ;dont insert between
bin_extend        db "/*.bin",0,0 ;here
template          db "%d",0
str_idb_Autoload  db "idb_Autoload",0
str_idb_Sleeptime db "idb_Sleeptime",0
```

```
.data?
```

```
hinst             HINSTANCE    ?    ; DLL instance
hwmain            HWND          ?    ; Handle of main OllyDbg window
textbuffer        db 512 dup(?)
byte_location     dd ?
SizeWritten       dd ?
hFileWrite        dd ?
hMemory_code      dd ?
pMemory_code      dd ?
SVWinClass        db 32 dup(?)
svthreadid        dd ?
sleep_time        dd ?
auto_load         dd ?
dw_buffer         dd ?
```

```
.code
```

; Entry point into a plug-in DLL. Many system calls require DLL instance which is passed to DllEntryPoint() as one of parameters. Remember it. Preferable way is to place initializations into ODBG\_Plugininit() and cleanup in ODBG\_Plugindestroy().

```
DllEntryPoint proc hi:HINSTANCE, reason:dword, res:dword
    .IF reason == DLL_PROCESS_ATTACH
        m2m         hinst, hi          ; Mark plugin instance
    .ENDIF
    return 1        ; Report success
DllEntryPoint endp
```

; ODBG\_Plugindata() is a "must" for valid OllyDbg plugin. It must fill in plugin name and return version of  
; plugin interface. If function is absent, or version is not compatible, plugin will be not installed. Short name  
; identifies it in the Plugins menu. This name is max. 31 alphanumerical

; characters or spaces + terminating '\0' long. To keep life easy for users,  
; this name should be descriptive and correlate with the name of DLL.

```
ODBG_Plugindata proc C shortname:ptr byte
    invoke Istrcpy, shortname, CTEXT("IsDebugPresent")    ; Name of plugin
    return PLUGIN_VERSION;
ODBG_Plugindata endp
```

; OllyDbg calls this obligatory function once during startup. Place all one-time initializations here. If all  
; resources are successfully allocated,

; function must return 0. On error, it must free partially allocated resources and return -1, in this case plugin  
; will be removed. Parameter ollydbgversion is the version of OllyDbg, use it to assure that it is compatible  
; with your plugin; hw is the handle of main OllyDbg window, keep it if necessary. Parameter features is  
; reserved for future extentions, do not use it.

```
ODBG_Plugininit proc C ollydbgversion:dword, hw:HWND, features:ptr dword
```

```
    ; Check that version of OllyDbg is correct.
```

```
    .IF ollydbgversion < PLUGIN_VERSION
```

```
        jmp    @@bad_exit
```

```
    .ENDIF
```

```
    invoke Addtolist, 0, 0, CTEXT("IsDebugPresent plugin v1.4 (SV 2oo3)")
```

```
    ; Keep handle of main OllyDbg window. This handle is necessary, for example,
```

```
    ; to display message box.
```

```
    m2m    hwmain, hw
```

```
    return 0
```

```
@@bad_exit:
```

```
    return -1
```

```
ODBG_Plugininit endp
```

; OllyDbg calls this optional function once on exit. At this moment, all MDI windows created by plugin are  
; already destroyed (and received WM\_DESTROY messages). Function must free all internally allocated  
; resources, like window classes, files, memory and so on.

```
ODBG_Plugindestroy proc C
```

```
    invoke Unregisterpluginclass,addr SVWinClass
```

```
    ret
```

```
ODBG_Plugindestroy endp
```

; Function is called when user opens new or restarts current application. Plugin should reset internal variables  
; and data structures to initial state.

ODBG\_Pluginreset proc C

```
invoke Pluginreadintfromini,hinst,addr str_idb_Autoload,0 ;Auto ?  
.if (eax!=0)  
    invoke Pluginreadintfromini,hinst,addr str_idb_Sleeptime,1000  
    mov sleep_time,eax ;save time value  
  
    lea eax,svthread  
    invoke CreateThread,NULL,NULL,eax,NULL,NULL,svthreadid  
.endif  
ret
```

svthread:

```
invoke Sleep,sleep_time ;Wait a little ;)  
invoke Pluggingetvalue,VAL_PROCESSID ;is something loaded ??  
.if (eax!=0)  
;    invoke SetWindowText,hwmain,CTEXT("OllyDbg -")  
  
    call get_byte_location  
    mov byte_location,eax  
    test eax,eax  
    jz svthread  
  
;1==Debugger 0==Clean ;)  
invoke Writememory,addr null_byte,byte_location,1,MM_RESTORE  
.if (eax!=1)  
    ;ooops  
    invoke Error,CTEXT("Error WriteMemory failed")  
.elseif  
    invoke Addtolist, 0, -1, CTEXT(" IsDebugPresent hidden")  
.endif  
invoke ExitThread,TRUE  
.endif
```

ODBG\_Pluginreset endp

```
; OllyDbg calls this optional function when user wants to terminate OllyDbg.
; All MDI windows created by plugins still exist. Function must return 0 if it is safe to terminate. Any non-zero
; return will stop closing sequence. Do not misuse this possibility! Always inform user about the reasons why
; termination is not good and ask for his decision!
```

```
ODBG_Pluginclose proc C
```

```
    ; For automatical restoring of open windows, mark in .ini file whether
    ; Bookmarks window is still open.
    return 0
```

```
ODBG_Pluginclose endp
```

```
; Function adds items either to main OllyDbg menu (origin=PM_MAIN) or to popup menu in one of standard
; OllyDbg windows. When plugin wants to add own menu items, it gathers menu pattern in data and returns 1,
; otherwise it must return 0. Except for static main menu, plugin must not add inactive items.
```

```
; Item indices must range in 0..63. Duplicated indices are explicitly allowed.
```

```
ODBG_Pluginmenu proc C uses ebx origin:dword, data:ptr byte, item:dword
```

```
    mov    eax, origin
    ; Menu creation is very simple. You just fill in data with menu pattern.
    ; Some examples:
    ; 0 Aaa,2 Bbb|3 Ccc|,, - linear menu with 3 items, relative IDs 0, 2 and
    ;                        3, separator between second and third item, last
    ;
    ;                        separator and commas are ignored;
    ; #A{0Aaa,B{1Bbb|2Ccc}} - unconditional separator, followed by popup menu
    ;                        A with two elements, second is popup with two
    ;                        elements and separator inbetween.
    .IF eax == PM_MAIN ; Plugin menu in main window
        invoke Istrcpy, data, CTEXT("0 &Hide,1 &Restore,2 &Option|3 &About|4 &Dumper")
        ;invoke Istrcpy, data, CTEXT("0 &Hide,1 &Restore|2 &About|3 &Dumper,4 &Initier")
        ; If your plugin is more than trivial, I also recommend to include Help.
        return 1
    .ENDIF
    return 0
ODBG_Pluginmenu endp
```

```
; This optional function receives commands from plugin menu in window of type origin. Argument action is
; menu identifier from ODBG_Pluginmenu(). If user activates automatically created entry in main menu, action
; is 0.
```

ODBG\_Pluginaction proc C origin:dword, action:dword, item:dword

```
mov    eax, origin
.IF eax == PM_MAIN
    mov    eax, action
    .IF !eax
        call get_byte_location
        mov byte_location,eax
        .if (eax)
            ;1==Debugger 0==Clean ;)
            invoke Writememory,addr null_byte,byte_location,1,MM_RESTORE
            .if (eax!=1)
                ;ooopps
                invoke Error,CTEXT("Error WriteMemory failed")
            .elseif
                invoke Addtolist, 0, -1, CTEXT(" IsDebugPresent hidden")
            .endif
        .endif
    .ELSEIF eax == 1
        call get_byte_location
        mov byte_location,eax
        .if (eax)
            ;resore original code
            invoke Writememory,addr the_byte,byte_location,1,MM_RESTORE
            .if (eax!=1)
                ;ooopps
                invoke Error,CTEXT("Error WriteMemory failed")
            .elseif
                invoke Addtolist, 0, -1, CTEXT(" IsDebugPresent restored")
            .endif
        .endif
    .ELSEIF eax == 2
        invoke DialogBoxParam, hinst, IDD_OPTION, hwmain, addr DlgOptionProc, NULL
    .ELSEIF eax == 3
        ; Menu item "About", displays plugin info.
        invoke MessageBox, hwmain, CTEXT("",13,10,"" 13,10," ",13,10," " ),\
            CTEXT("IsDebuggerPresent plugin"),MB_OK or MB_ICONINFORMATION
    .ELSEIF eax == 4
        invoke DialogBoxParam, hinst, IDD_DUMP, hwmain, addr DlgDumpProc, NULL
    .ENDIF
.ENDIF
ret
```

```

ODBG_Pluginaction endp
get_byte_location proc
    push ebx
    invoke Getcputhreadid
    .if (eax)
        invoke Findthread,eax           ;retrive thread info
        assume eax:ptr t_thread
        push [eax].reg.base[4*4]       ;base of FS
        pop ebx
        add ebx,30h
        invoke Readmemory,addr dw_buffer,ebx,4,MM_RESTORE
        mov eax,dw_buffer
        add eax,2h
    .endif
    pop ebx
    ret

get_byte_location endp
DlgDumpProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
LOCAL dump_start:DWORD
LOCAL dump_size:DWORD

    .IF uMsg == WM_INITDIALOG
        mov ofn.lStructSize,SIZEOF ofn
        push hWnd
        pop ofn.hWndOwner
        push hinst
        pop ofn.hInstance
        mov ofn.nMaxFile,MAXSIZE
    .ELSEIF uMsg == WM_CLOSE
        invoke SendMessage, hWnd, WM_COMMAND, IDC_BTCANCEL, 0
    .ELSEIF uMsg==WM_COMMAND
        mov eax,wParam
        mov edx,wParam
        shr edx,16
        .IF dx==BN_CLICKED
            .IF ax==IDC_BTCANCEL
                invoke EndDialog, hWnd, NULL
            .ELSEIF ax==IDC_BTDDUMP
                invoke Plugingetvalue,VAL_PROCESSID    ;is something loaded ??
                .if (eax!=0)
                    pushad
                    invoke GetDlgItemText,hWnd,IDC_EDTOFFSET,addr textbuffer,10
                    invoke htodw,addr textbuffer
                    mov dump_start,eax
                    invoke GetDlgItemText,hWnd,IDC_EDTSIZE,addr textbuffer,10
                .endif
            .endif
        .endif
    .endif

```

```

invoke htdw,addr textbuffer
mov dump_size,eax

.if dump_start!=0 && dump_size!=0
;Alloc Mem
invoke GlobalAlloc,GMEM_MOVEABLE or GMEM_ZEROINIT,dump_size
mov hMemory_code,eax
invoke GlobalLock,hMemory_code
mov pMemory_code,eax
;Read in Mem
invoke Readmemory,pMemory_code,dump_start,dump_size,MM_RESTORE
.if (eax==dump_size)
    push hWnd
    pop ofn.hWndOwner
    mov ofn.Flags,OFN_OVERWRITEPROMPT
    mov ofn.lpstrFilter, OFFSET FilterString
    mov ofn.lpstrDefExt, OFFSET bin_extend
    mov ofn.lpstrFile, OFFSET textbuffer
    mov ofn.nMaxFile,MAXSIZE
    mov [textbuffer],0 ;filename buffer a vide
        invoke GetSaveFileName, ADDR ofn
    .if eax==TRUE
        invoke CreateFile,ADDR textbuffer,\
        GENERIC_READ or GENERIC_WRITE ,\
        FILE_SHARE_READ or FILE_SHARE_WRITE,\
        NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,\
        NULL
        mov hFileWrite,eax
        ;Save mem in file
        invoke WriteFile,hFileWrite,pMemory_code,dump_size,ADDR SizeWritten,NULL
        invoke CloseHandle,hFileWrite
        invoke Flash,CTEXT("File successfully writted !!")
        .endif
        .endif
        ;Free Mem
        invoke GlobalUnlock,pMemory_code
        invoke GlobalFree,hMemory_code
    .endif
    popad
.else
    invoke Error,CTEXT("Error Nothing loaded")
.endif
.ENDIF
.ENDIF
.ELSE

```

```
    mov eax, FALSE
    ret
.ENDIF
```

```
mov eax, TRUE
ret
DlgDumpProc endp
```

```
DlgOptionProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
```

```
.IF uMsg == WM_INITDIALOG
;read params from ini & update windows
invoke Pluginreadintfromini,hinst,addr str_idb_Autoload,0
mov auto_load,eax
invoke Pluginreadintfromini,hinst,addr str_idb_Sleeptime,1000
mov sleep_time,eax
invoke wsprintf,addr textbuffer,offset template,sleep_time
invoke SetDlgItemText,hWnd,IDC_SLEEPTIME,addr textbuffer
.if (auto_load!=0)
    invoke GetDlgItem,hWnd,IDC_CHK1
    invoke SendMessage,eax,BM_SETCHECK,BST_CHECKED,0
.endif
.ELSEIF uMsg == WM_CLOSE
    invoke SendMessage, hWnd, WM_COMMAND, IDC_BTNCANCEL, 0
.ELSEIF uMsg==WM_COMMAND
    mov eax,wParam
    mov edx,wParam
    shr edx,16
    .IF dx==BN_CLICKED
        .IF ax==IDC_BTN2 ;CANCEL
            invoke EndDialog, hWnd, NULL
        .ELSEIF ax==IDC_BTN1 ;SAVE
            invoke GetDlgItemText,hWnd,IDC_SLEEPTIME,addr textbuffer,10
            invoke atodw,addr textbuffer
            mov sleep_time,eax
            ;save params to ini
            invoke Pluginwriteinttoini,hinst,addr str_idb_Sleeptime,sleep_time
            invoke GetDlgItem,hWnd,IDC_CHK1
            invoke SendMessage,eax,BM_GETSTATE,0,0
            .if eax==BST_CHECKED
                invoke Pluginwriteinttoini,hinst,addr str_idb_Autoload,1
            .elseif
                invoke Pluginwriteinttoini,hinst,addr str_idb_Autoload,0
            .endif
            invoke Flash,CTEXT("Option saved !!")
        .ENDIF
    .ENDIF
```

```

.ENDIF
.ELSE

    mov eax, FALSE
    ret
.ENDIF
    mov eax, TRUE
    ret
DlgOptionProc endp

end DllEntryPoint

```

---

## 14<sup>th</sup> Mode – Anti Ring 0 Debug – IsDebuggerPresent() – Advanced Anti-Debugging

---

Code:

```

.386
.model flat,stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
include \masm32\include\advapi32.inc
includelib \masm32\lib\advapi32.lib

.data
forma          db "GlobalFlag in fs:[30]+68 is equal to %08x",13,10,"GlobalFlag in registry is equal to %08x",0
forma1         db "GlobalFlag in fs:[30]+68 is equal to %08x",13,10,"GlobalFlag in registry is equal to %08x",0
tite           db "you are not running inside debugger",0
tite1          db " you are running this under debugger",0
subkeyname     db "SYSTEM\CURRENTCONTROLSET\CONTROL\SESSION MANAGER",0
valuenam      db "GLOBALFLAG",0

.DATA?
buffer        db 120h dup (?)
buffer1       db 120h dup (?)
buffer2       db 120h dup (?)
buffer3       db 120h dup (?)
buffer4       dd ?

```

```

.CODE
start:
    mov buffer4,45h
    invoke RegOpenKeyEx,HKEY_LOCAL_MACHINE,ADDR subkeyname,NULL,KEY_ALL_ACCESS,addr buffer1
    invoke RegQueryValueEx,dword ptr ds:[buffer1],addr valuenam, NULL,addr buffer2,addr buffer3,addr
buffer4
    assume fs:nothing
    mov eax,fs:[30h]
    mov eax,[eax+68h]
    .if dword ptr ds:[buffer3]==eax
        invoke wsprintf,addr buffer,addr forma1,eax,dword ptr ds:[buffer3]
        invoke MessageBox,NULL,offset buffer,offset tite,NULL
    .elseif
        invoke wsprintf,addr buffer,addr forma1,eax,dword ptr ds:[buffer3]
        invoke MessageBox,NULL,offset buffer,offset tite1,NULL
    .endif
    invoke RegCloseKey,dword ptr ds:[buffer1]
    invoke ExitProcess,NULL
end start

```

---

## 15<sup>th</sup> Mode – Anti Ring 0 Debug – IsDebuggerPresent() – Advanced Anti-Debugging – Olly

---

```

.586
.model flat, stdcall
option casemap:none

include c:\masm32\INCLUDE\Windows.inc
include c:\masm32\INCLUDE\user32.inc
include c:\masm32\INCLUDE\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.data
strCaption    db          "OllyDbg Detector!",0
strFound     db          "OllyDbg found!",0
strNotFound  db          "OllyDbg NOT found!",0
strOllyDbg   db          "OLLYDBG.EXE",0h

valCurrentPiD dd 0
valParentPiD  dd 0

hSnapShot    dd 0

```

```

.data?
proces          PROCESSENTRY32 <>

.code
start:

; Create the snapshot
invoke CreateToolhelp32Snapshot, TH32CS_SNAPPROCESS,NULL
mov hSnapShot,eax

;Get the ProcessId of the Current Process
invoke GetCurrentProcessId
mov valCurrentPiD,eax

lea esi,offset proces
assume esi:ptr PROCESSENTRY32
mov [esi].dwSize,sizeof PROCESSENTRY32

;Begin the first Loop and find the current process

;using the valCurrentPiD
invoke Process32First,hSnapShot,addr proces

lea esi,offset proces
assume esi:ptr PROCESSENTRY32
mov ebx,valCurrentPiD

cmp ebx,[esi].th32ProcessID
jne nope1

nope1:
invoke Process32Next,hSnapShot,addr proces

lea esi,offset proces
assume esi:ptr PROCESSENTRY32

mov ebx,valCurrentPiD
cmp ebx,[esi].th32ProcessID
jne nope1

push [esi].th32ParentProcessID
pop valParentPiD
invoke CloseHandle,hSnapShot

```

```

; Create the snapshot again
    invoke CreateToolhelp32Snapshot, TH32CS_SNAPPROCESS,NULL
    mov hSnapShot,eax

    mov [esi].dwSize,sizeof PROCESSENTRY32

; Begin the second Loop and find the parent

; process using the valParentPiD
    invoke Process32First,hSnapShot,addr proces

    lea esi,offset proces
    assume esi:ptr PROCESSENTRY32
    mov ebx,valParentPiD
    cmp ebx,[esi].th32ProcessID
    jne nope2

nope2:
    invoke Process32Next,hSnapShot,addr proces
    lea esi,offset proces
    assume esi:ptr PROCESSENTRY32
    mov ebx,valParentPiD
    cmp ebx,[esi].th32ProcessID
    jne nope2

; Extract filename of the Parent Process from the whole string
    lea eax, [esi].szExeFile
    push eax
    invoke lstrlen,eax
    sub eax,11
    pop ebx
    add ebx,eax

; Case Upper the string and compare it with "OLLYDBG.EXE"
    invoke CharUpper,ebx
    invoke lstrcmp,ebx,addr strOllyDbg

    .IF eax==0
        invoke MessageBox,0,addr strFound,addr strCaption,0
    .ELSE
        invoke MessageBox,0,addr strNotFound,addr strCaption,0
    .ENDIF
    invoke CloseHandle,hSnapShot
    invoke ExitProcess,0
end start

```

---

## END NOTE

Invisible KERNEL hooks will hide all the possible Kernel and Marshall (NTDLL) Hooks from xAurora and vice versa. That's why you may never been able to see any KMD(s) running in the browser, even when you loaded browser in to any debugger. Also, I have included many Anti-Debugging techniques, Anti-Debug Attacking Methods and many other ultra secured Anti-Reverse Engineering methodologies to achieve the maximum protection for the xAurora. And that information never gave out to any personal till now. All the software anti-reverse engineering tricks coded using Microsoft Macro Assembler 8.2x. And also I have used most of custom made API(s) and NT Undocumented API(s) for better protection of the browser. Above all the examples has been successfully redesigned, recoded (rewritten) and included into xAurora.

## SPECIAL NOTE

I am sending the unwrapped (Protection Removed) xAurora Main Executable file to Mr. Gotaimbara to do the Live Demo and PoC of KMD Availability in the browser.

## CONCLUSION

Mr. Anonymous Skywalker, Mr. Harshadeva Ariyasinghe and Mr. Kalinga Athulathmudali's reverse engineering methodologies are very lamer and they have very basic NT Kernel/Low Level knowledge. Also they may never work with UNDOCUMENTED NT API(s) in NT Subsystem. xAurora is based on Windows Kernel Mode Drivers. This guideline document will help you to understand why Mr. Anonymous Skywalker, Mr. Harshadeva Ariyasinghe and Mr. Kalinga Athulathmudali xAurora Kernel Mode PoC failed and they couldn't prove it. I know my own code better than all of you. Because, I am the founder of xAurora's concepts and I am the programmer of the xAurora Web Browser. No one can admit the wrong conclusion without proving it in the real world and to the community. Because, xAurora is a COPYRIGHTED, TRADEMARKED and PATENTED SOFTWARE.

xAurora is a great Sri Lankan product that entirely coded in Win32 Assembly Language. Hope you may be able to understand it. In future I will show you many stories behind the xAurora case. Hope Mr. Gotaimbara will help me to sort out the matters soon. Thank you very much for the great support and your support in the future is greatly appreciated.

Kind Regards

Dr. Sameera de Alwis

Founder – Team xAurora 2009