Advanced Malware Analysis Training Series

Anti-Analysis Techniques

Swapnil Pathak

www.SecurityXploded.com

Disclaimer

Acknowledgement

- Special thanks to **Null** community for their extended support and co-operation.
- Special thanks to **ThoughtWorks** for the beautiful venue.
- Thanks to all the trainers who have devoted their precious time and countless hours to make it happen.

Advanced Malware Analysis Training

This presentation is part of our **Advanced Malware Analysis** Training program. Currently it is delivered only during our local meets for FREE of cost.

For complete details of this course, visit our Security Training page.

Who am I?

Swapnil Pathak

- Security Researcher
- Reversing, Malware Analysis, Exploit Analysis etc.
- E-mail: swapnilpathak101@gmail.com

Agenda

- Introduction
- Anti-Reversing techniques
  - Anti-Debugging
  - Anti-VM
- Anti-Anti-Reversing techniques
- Q & A

Anti-Reversing

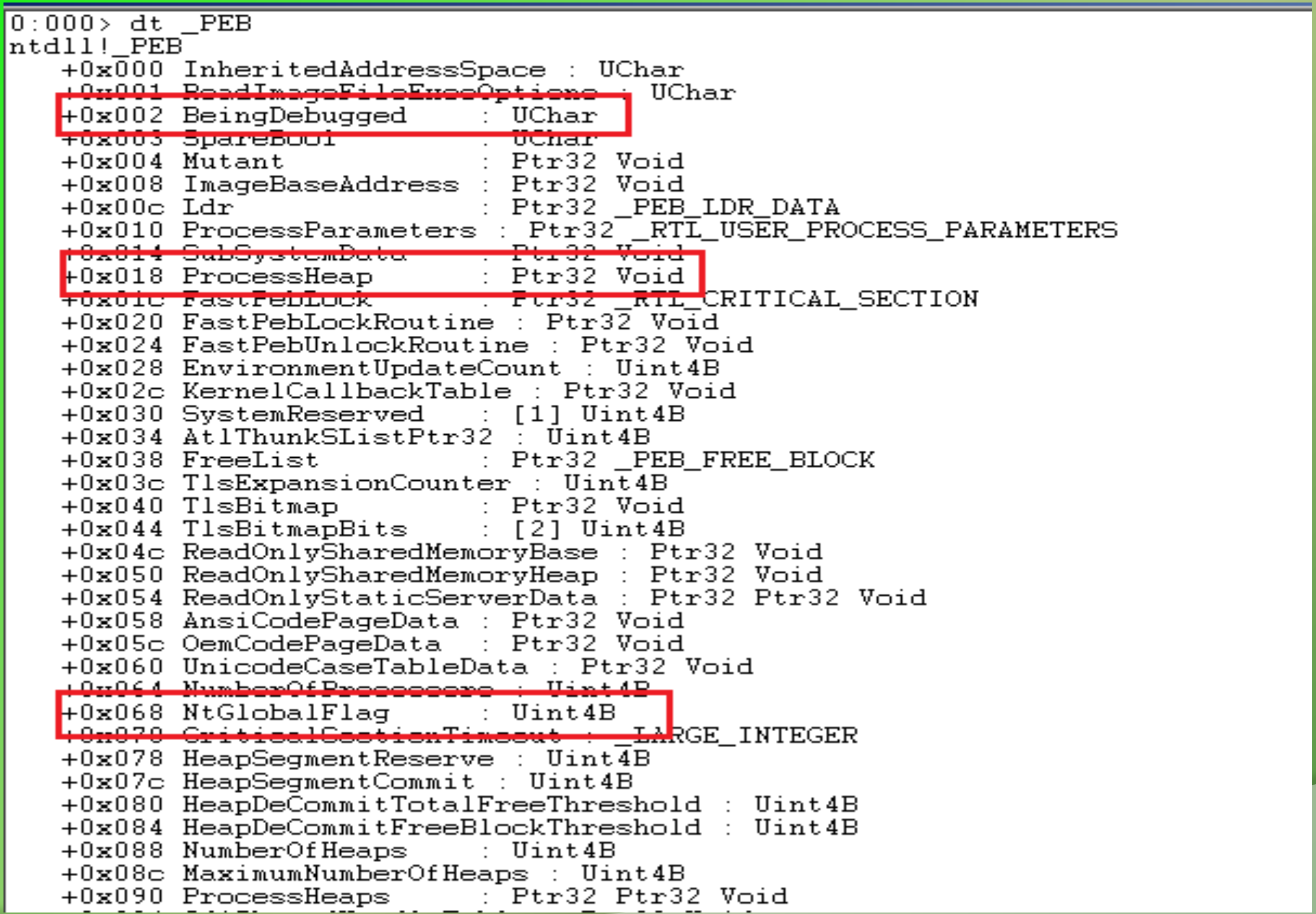
- Implementation of techniques in code to hinder attempts at reverse engineering or debugging a target binary.
- Used by commercial protectors, packers and malicious software
- Covers
  - Anti-Debugging
  - Anti-VM
  - Anti-Disassembly
  - Code Obfuscation

Anti-Debug Techniques

- Techniques implemented to detect if the program is running under control of a debugger.
- Categorized as below
  - API Based
  - Flags Based
  - Timing Based
  - Exception Based
  - Breakpoint Detection

PEB (Process Environment Block)

- Structure maintained by OS for each running process
- Contains user mode parameters associated with a process
- Including loaded modules list, debugger status etc
- Referenced through fs:[30h]
- !peb command in Windbg



```
0:000> dt _PEB
ntdll!_PEB
   +0x000 InheritedAddressSpace : UChar
   +0x001 ReadImageFileExecOptions : UChar
   +0x002 BeingDebugged    : UChar
   +0x003 SpareBool        : UChar
   +0x004 Mutant           : Ptr32 Void
   +0x008 ImageBaseAddress : Ptr32 Void
   +0x00c Ldr              : Ptr32 _PEB_LDR_DATA
   +0x010 ProcessParameters : Ptr32 _RTL_USER_PROCESS_PARAMETERS
   +0x014 SubSystemData    : Ptr32 Void
   +0x018 ProcessHeap      : Ptr32 Void
   +0x01c FastPebLock      : Ptr32 _RTL_CRITICAL_SECTION
   +0x020 FastPebLockRoutine : Ptr32 Void
   +0x024 FastPebUnlockRoutine : Ptr32 Void
   +0x028 EnvironmentUpdateCount : Uint4B
   +0x02c KernelCallbackTable : Ptr32 Void
   +0x030 SystemReserved   : [1] Uint4B
   +0x034 AtlThunkSListPtr32 : Uint4B
   +0x038 FreeList         : Ptr32 _PEB_FREE_BLOCK
   +0x03c TlsExpansionCounter : Uint4B
   +0x040 TlsBitmap        : Ptr32 Void
   +0x044 TlsBitmapBits    : [2] Uint4B
   +0x04c ReadOnlySharedMemoryBase : Ptr32 Void
   +0x050 ReadOnlySharedMemoryHeap : Ptr32 Void
   +0x054 ReadOnlyStaticServerData : Ptr32 Ptr32 Void
   +0x058 AnsiCodePageData : Ptr32 Void
   +0x05c OemCodePageData  : Ptr32 Void
   +0x060 UnicodeCaseTableData : Ptr32 Void
   +0x064 NumberOfProcessors : Uint4B
   +0x068 NtGlobalFlag     : Uint4B
   +0x070 CriticalSectionTimeout : _LARGE_INTEGER
   +0x078 HeapSegmentReserve : Uint4B
   +0x07c HeapSegmentCommit : Uint4B
   +0x080 HeapDeCommitTotalFreeThreshold : Uint4B
   +0x084 HeapDeCommitFreeBlockThreshold : Uint4B
   +0x088 NumberOfHeaps    : Uint4B
   +0x08c MaximumNumberOfHeaps : Uint4B
   +0x090 ProcessHeaps     : Ptr32 Ptr32 Void
```

API Based

- IsDebuggerPresent
  - Exported by kernel32.dll
  - Function accepts no parameters
  - Checks if BeingDebugged flag in Process Environment Block (PEB) is set
  - Returns 1 if process is being debugged, 0 otherwise

```
call IsDebuggerPresent
test eax,eax
jnz debugger_detected
```

CheckRemoteDebuggerPresent => NtQueryInformationProcess

- CheckRemoteDubggerPresent(Handle to the target process, Pointer to a variable
- Sets the variable to TRUE if the specified process is being debugged or FALSE otherwise

```
push offset dbg
push -1
call CheckRemoteDebuggerPresent
test eax, eax
jne debugger_detected
```

API Based

- NtQueryInformationProcess => ZwQueryInformationProcess
  - Exported by ntdll.dll
  - Retrieves information about the specified process.
  - NtQueryInformationProcess(ProcessHandle, ProcessInformationClass, ProcessInformation, ProcessInformationLength, ReturnLength)
  - ProcessHandle – Handle to the process for which information is to be retrieved.
  - ProcessInformationClass : Type of process information to be retrieved.
    - Accepts following values : **ProcessBasicInformation(0x0), ProcessDebugPort(0x07), ProcessWow64Information(0x26), ProcessImageFileName(0x27)**
  - ProcessDebugPort : Retrieves port number of the debugger for the process.
  - Non Zero value indicates that the process is being debugged.

API Based

- OutputDebugString
  - Exported by kernel32.dll
  - Accepts one parameter : Null terminated string to be displayed
  - Sends a string to the debugger for display
  - Sends an error if there is no active debugger for the process to receive the string.
  - No error indicates presence of a debugger.
- FindWindow
  - Exported by user32.dll
  - Used to search windows by name or class.
  - Detect debugger with graphical user interface

API Based

- CloseHandle
  - Exported by kernel32.dll
  - Involves passing invalid handle
  - If debugger present EXCEPTION_INVALID_HANDLE (0xC0000008) will be raised
  - Above exception if intercepted by an exception handler, indicates presence of a debugger.

```
push 12ab ; any illegal value

call CloseHandle
```

Flags Based

- BeingDebugged Flag
  - Present in PEB (Process Environment Block) at offset 0x2
  - Set to 1 if the process is being debugged.

```
mov eax, dword [fs:0x30]

movzx eax, byte [eax + 0x02] ; PEB.BeingDebugged

test eax, eax

jnz debugger_detected
```

Flags Based

- NTGlobal Flag
  - Present in PEB (Process Environment Block) at offset
    - 0x68 - 32 bit systems
    - 0xBC - 64 bit systems
  - Contains value 0x70 if the process is being debugged.
  - FLG_HEAP_ENABLE_TAIL_CHECK(0x10), FLG_HEAP_ENABLE_FREE_CHECK(0x20), FLG_HEAP_VALIDATE_PARAMETERS(0x40)

```
mov eax, fs :[30h]
mov eax, [eax+68h]
and eax, 0x70
test eax, eax
jne debugger_detected
```

Flags Based

- Heap Flags
  - ProcessHeap present in PEB (Process Environment Block) at offset 0x18
  - Has Flags and ForceFlags Fields set to 0x02 (HEAP_GROWABLE) and 0x0 respectively if the process is not being debugged.

```
mov eax, fs:[30h]
mov eax, [eax + 18h] ; eax = PEB.ProcessHeap
cmp [eax + 10h] , 0 ; ProcessHeap.ForceFlags
jne debugger_detected
cmp [eax + 0x0c], 2 ; ProcessHeap.Flags
jne debugger_detected
```

Timing Based

- Timing Checks
  - Compares time spent executing instructions normally and while being debugged.
  - Longer time taken compared to normal run indicates that the binary is being debugged.
  - RDTSC (Read Time Stamp Counter)
  - GetTickCount()
  - QueryPerformanceCounter()

```
rdtsc
mov eax, ebx
…….
rdtsc
sub eax, ecx
cmp eax, 0x100
ja debugger_detected
```

Exception Based

- Interrupts
  - Consists of inserting interrupt instructions in middle of valid sequence of instructions.
  - INT3 breakpoint (0xCC, 0xCD 0x03)
  - INT 1 single step

    INT2D are stepped through inside the debugger, exception is raised.
  - If the process is being debugged, exception handler is not invoked as the debugger typically handles the exception
  - Checks such as setting of flags inside exception handler are used to detect presence of the debugger.

Breakpoint Detection

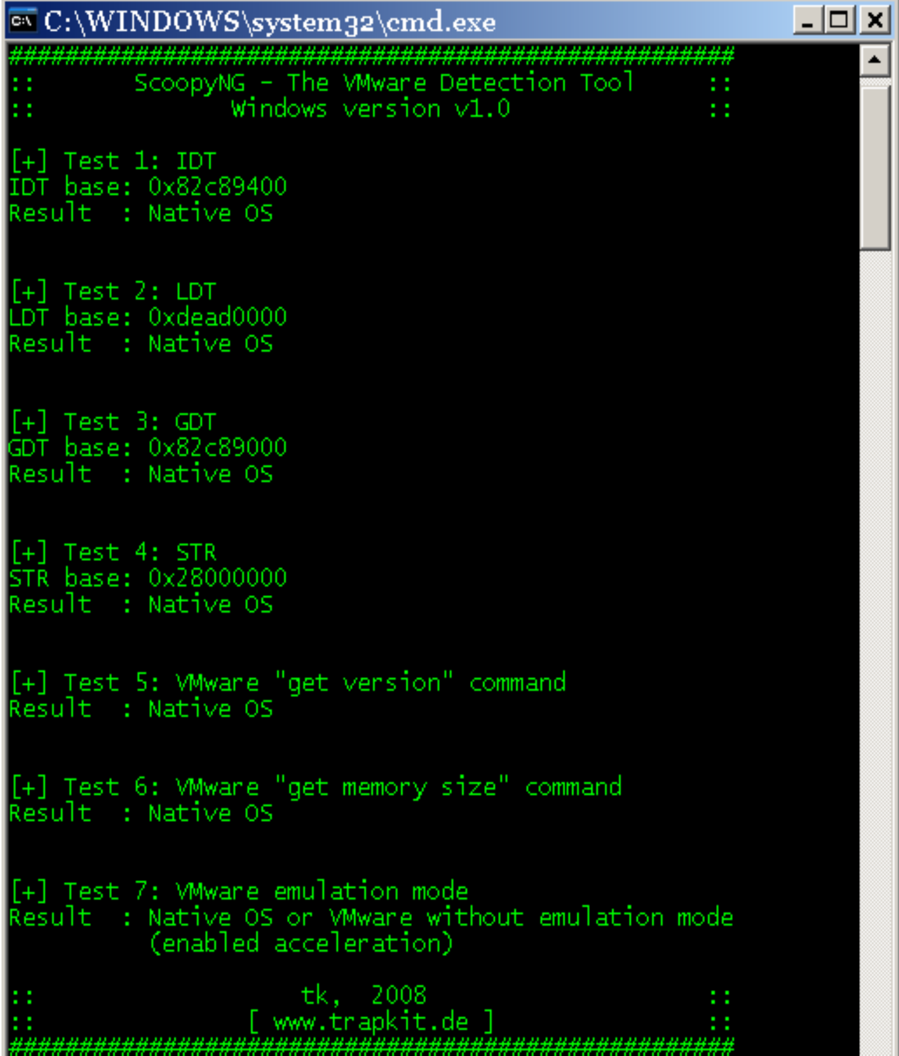
- Hardware Breakpoints
  - Whenever an exception occurs, a context structure is created and passed to the exception handler.
  - Context structure contains values of general registers, control registers, debug registers.
  - Binary being debugged with hardware breakpoints in use will contain values in debug registers indicating presence of debugger.
- Memory Breakpoints
  - Implemented using Guard pages.
  - Guard Pages are set using the PAGE_GUARD page protection modifier.
  - Address accessed part of Guard Page will result in STATUS_GUARD_PAGE_VIOLATION exception.
  - Process being debugged under Ollydbg will treat this as a memory breakpoint and no exception will be raised.

VM Detection

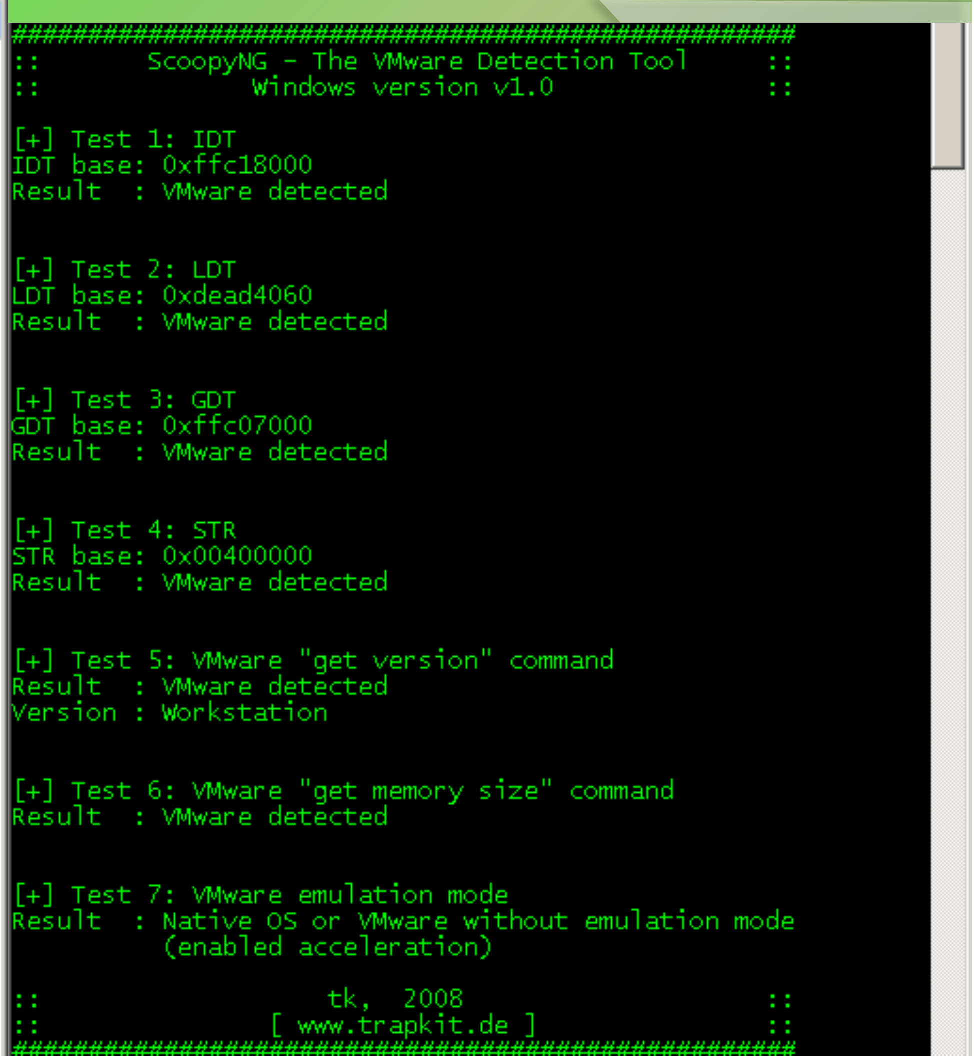
- Techniques implemented to detect if the binary is being executed in a virtual environment.
- Techniques include
  - Memory based
  - Backdoor I/O communication port
  - Process/Registry

Techniques

- Memory specific techniques include Red Pill
  - Only one IDT, GDT, LDT per processor.
  - IDT : Used by OS to determine correct response to interrupts and exceptions
  - GDT/LDT : Define characteristics of the various memory areas used during program execution such as base address, size and access privileges.
  - IDTR, GDTR, LDTR are internal registers that store the address of these respective tables.
  - To avoid conflicts between host and guest OS , virtual machine needs to relocate IDT, GDT, LDT
  - SIDT, SGDT and SLDT are instructions to retrieve values from IDTR, GDTR and LDTR respectively.
  - Base address stored in the register define if under Virtualized environment.
  - IDT is at 0x80ffffff in Windows, 0xE8xxxxxx in VirtualPC, 0xFFxxxxxx in Vmware.



```
C:\WINDOWS\system32\cmd.exe
####################################################
::      ScoopyNG - The VMware Detection Tool      ::
::              Windows version v1.0               ::

[+] Test 1: IDT
IDT base: 0x82c89400
Result  : Native OS


[+] Test 2: LDT
LDT base: 0xdead0000
Result  : Native OS


[+] Test 3: GDT
GDT base: 0x82c89000
Result  : Native OS


[+] Test 4: STR
STR base: 0x28000000
Result  : Native OS


[+] Test 5: VMware "get version" command
Result  : Native OS


[+] Test 6: VMware "get memory size" command
Result  : Native OS


[+] Test 7: VMware emulation mode
Result  : Native OS or VMware without emulation mode
          (enabled acceleration)

::                   tk,  2008                   ::
::               [ www.trapkit.de ]               ::
####################################################
```



```
####################################################
::      ScoopyNG - The VMware Detection Tool      ::
::              Windows version v1.0               ::

[+] Test 1: IDT
IDT base: 0xffc18000
Result  : VMware detected


[+] Test 2: LDT
LDT base: 0xdead4060
Result  : VMware detected


[+] Test 3: GDT
GDT base: 0xffc07000
Result  : VMware detected


[+] Test 4: STR
STR base: 0x00400000
Result  : VMware detected


[+] Test 5: VMware "get version" command
Result  : VMware detected
Version : Workstation


[+] Test 6: VMware "get memory size" command
Result  : VMware detected


[+] Test 7: VMware emulation mode
Result  : Native OS or VMware without emulation mode
          (enabled acceleration)

::                   tk,  2008                   ::
::               [ www.trapkit.de ]               ::
####################################################
```

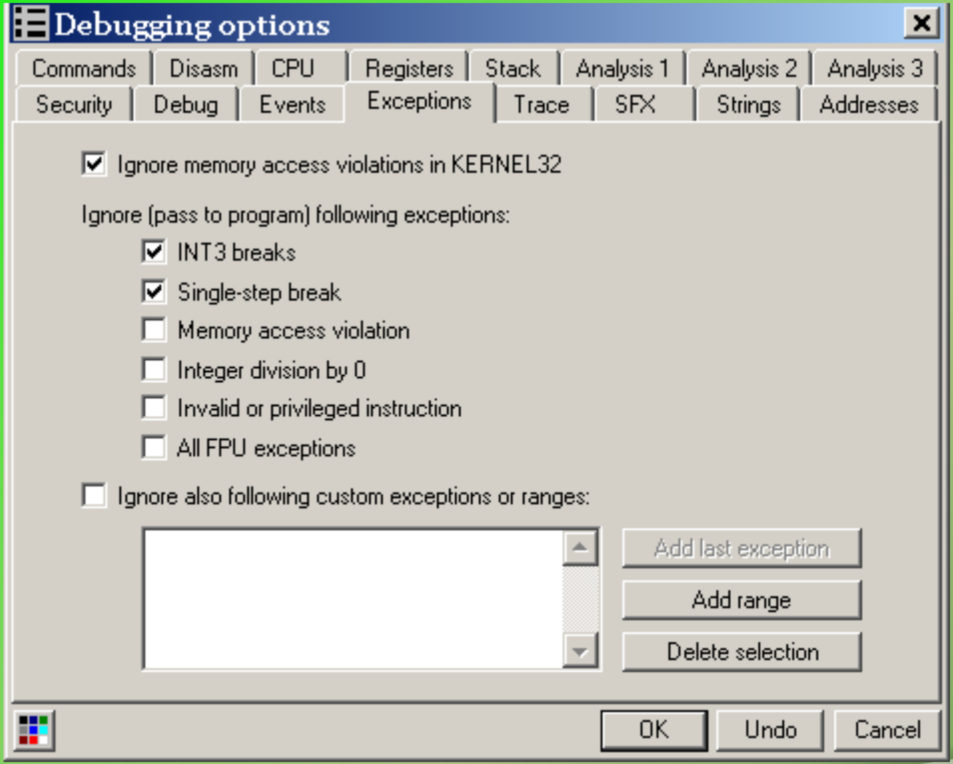
Techniques

- Backdoor I/O port
  - OS running inside a VMWware uses port name ‘VX’ for communication between the host and guest OS.
  - Data from this port can be read using various opcodes using “IN” instruction
    - 0x0A provides Vmware version.
    - 0x14 gets the memory size.
    - Value checked against ‘VMXh’ to detect presence of Vmware
- Process/Service Check
  - Check for Vmware related process, services
  - Process Name (Associated Service Name)
  - vmacthlp.exe (Vmware Physical Disk Helper Service)
  - vmtoolsd.exe (Vmware Tools)

Anti-Anti-Debug Techniques

- Manually patch values in memory, registers, return values from APIs
- NOP sequence of instructions
- Use of Plugins
  - OllyAdvanced
  - HideDebugger
  - Phantom
  - Anti-Anti-Debugger Plugins : https://code.google.com/p/aadp/

OllyDbg Screenshots

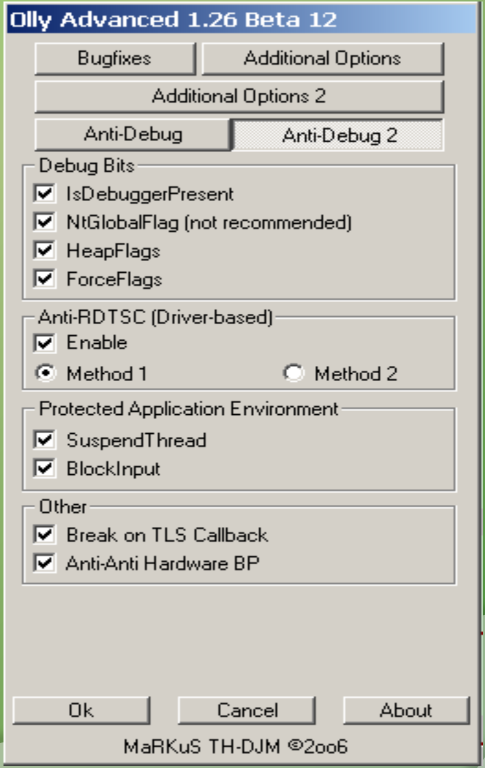


[✓] Ignore memory access violations in KERNEL32

Ignore (pass to program) following exceptions:

- [✓] INT3 breaks
- [✓] Single-step break
- [ ] Memory access violation
- [ ] Integer division by 0
- [ ] Invalid or privileged instruction
- [ ] All FPU exceptions

[ ] Ignore also following custom exceptions or ranges:



Debug Bits

- [✓] IsDebuggerPresent
- [✓] NtGlobalFlag (not recommended)
- [✓] HeapFlags
- [✓] ForceFlags

Anti-RDTSC (Driver-based)

- [✓] Enable
- [✓ Method 1] Method 2

Protected Application Environment

- [✓] SuspendThread
- [✓] BlockInput

Other

- [✓] Break on TLS Callback
- [✓] Anti-Anti Hardware BP

References

Complete Reference Guide for Advanced Malware Analysis Training

[Include links for all the Demos & Tools]

Thank You !

www.SecurityXploded.com