IBM Global Services

The Art of Unpacking

Mark Vincent Yason
Malcode Analyst
X-Force Research & Development
myason@us.ibm.com

IBM Internet Security Systems
Ahead of the threat.™

The Art Of Unpacking

- Packers are one of the most interesting puzzles to solve in the Reverse Engineering field
- Packers are created to protect legitimate applications, but they are also used by malcode
- Overtime, new anti-reversing techniques are integrated into packers
- Meanwhile, researchers on the other side of the fence find ways to break/bypass these protections... it is a mind game
- Anti-reversing techniques are also interesting because a lot of knowledge about Windows internals are gained

The Art Of Unpacking

- This talk focuses on commonly used and interesting anti-reversing techniques employed by packers
- Also discusses ways on how to bypass/disable anti-reversing techniques/tricks
- This talk aims to share information to researchers, reversers and malcode analysts
- Information presented can be used in identifying and solving anti-reversing tricks employed packed malicious code

Anti-Reversing Topics

- Debugger Detection
- Breakpoint and Patching Detection
- Anti-Analysis
- Advanced and Other Techniques
- Tools

IBM Global Services

The Art Of Unpacking
Debugger Detection

IBM Internet Security Systems
Ahead of the threat.™

Debugger Detection > PEB.BeingDebugged Flag

- Most basic (and obvious) debugger detection technique
- PEB.BeingDebugged flag is 1 if process is being debugged, 0 if not
- fs:[0x30] points to the PEB
- kernel32!IsDebuggerPresent() checks this flag
- Packers may obfuscate the check since it is very obvious

```
lkd> dt _PEB
   +0x000 InheritedAddressSpace    : UChar
   +0x001 ReadImageFileExecOptions : UChar
   +0x002 BeingDebugged            : UChar
   :::
```

Debugger Detection > PEB.BeingDebugged Flag

- Example: Using IsDebuggerPresent() and directly checking PEB.BeingDebugged

```
    ; 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
```

- Solution:
  - Easily bypassed by patching PEB.BeingDebugged flag with 0
  - Ollyscript “dbh” command patches this flag
  - Olly Advanced also has an option to patch this flag

Debugger Detection > PEB.NtGlobalFlag, Heap.HeapFlags, Heap.ForceFlags

- PEB.NtGlobalFlag contains the value 0x0 if process is not debugged, usually 0x70 if debugged

```
lkd> dt _PEB
   :::
   +0x068 NtGlobalFlag     : Uint4B
   :::
```

- The following flags are set if process is being debugged:
  - FLG_HEAP_ENABLE_TAIL_CHECK (0x10)
  - FLG_HEAP_ENABLE_FREE_CHECK (0x20)
  - FLG_HEAP_VALIDATE_PARAMETERS (0x40)
- Flags can overridden via registry setting or gflags.exe

Debugger Detection > PEB.NtGlobalFlag, Heap.HeapFlags, Heap.ForceFlags

- Because NtGlobalFlags are set, Heap Flags will also be set

```
lkd> dt _HEAP
   :::
   +0x00c Flags            : Uint4B
   +0x010 ForceFlags       : Uint4B
   :::
```

- Heap.Flags is 0x2 (HEAP_GROWABLE) if process is not debugged, usually 0x50000062 if debugged (depending on NtGlobalFlags)
  - HEAP_TAIL_CHECKING_ENABLED (0x20)
  - HEAP_FREE_CHECKING_ENABLED (0x40)
- Heap.ForceFlags is 0x0 if process is not debugged, usually, 0x40000060 if debugged (Flags & 0x6001007D)

Debugger Detection > PEB.NtGlobalFlag, Heap.HeapFlags, Heap.ForceFlags

- Example: Checks PEB.NtGlobalFlags and flags of 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],0x2
jne     .debugger_found

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

Debugger Detection > PEB.NtGlobalFlag, Heap.HeapFlags, Heap.ForceFlags

- Solution:
  - Patch NtGlobalFlag, PEB.ProcessHeap Flags
  - Olly Advanced plug-in or Ollyscript:

```
var     peb
var     patch_addr
var     process_heap

//retrieve PEB via a hardcoded TEB address (first thread: 0x7ffde000)
mov     peb,[7ffde000+30]

//patch PEB.NtGlobalFlag
lea     patch_addr,[peb+68]
mov     [patch_addr],0

//patch PEB.ProcessHeap.Flags/ForceFlags
mov     process_heap,[peb+18]
lea     patch_addr,[process_heap+0c]
mov     [patch_addr],2
lea     patch_addr,[process_heap+10]
mov     [patch_addr],0
```

Debugger Detection > DebugPort

- DebugPort field of the EPROCESS kernel structure is 0 if process is not being debugged, otherwise, it contains a non-zero value
- ntdll!NtQueryInformationProcess (ProcessDebugPort) queries the DebugPort field, returns 0xFFFFFFFF if DebugPort is non-zero, otherwise returns 0
- kernel32!CheckRemoteDebuggerPresent() uses ntdll!NtQueryInformationProcess () to check if the process is being debugged

```
BOOL CheckRemoteDebuggerPresent(
  HANDLE hProcess,
  PBOOL  pbDebuggerPresent
)
```

Debugger Detection > DebugPort

- Example: Using CheckRemoteDebuggerPresent() and NtQueryInformationProcess()

```
; 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
```

Debugger Detection > DebugPort

- Solution: Manipulating return value of NtQueryInformationProcess (ollyscript sample)

```
    // set a breakpoint handler
    eob     bp_handler_NtQueryInformationProcess
    // set a breakpoint where NtQueryInformationProcess returns
    gpa     "NtQueryInformationProcess", "ntdll.dll"
    find    $RESULT, #C21400#   //retn 14
    mov     bp_NtQueryInformationProcess,$RESULT
    bphws   bp_NtQueryInformationProcess,"x"
    run

bp_handler_NtQueryInformationProcess:
    //ProcessInformationClass == ProcessDebugPort?
    cmp     [esp+8], 7
    jne     bp_handler_NtQueryInformationProcess_continue
    //patch ProcessInformation to 0
    mov     patch_addr, [esp+c]
    mov     [patch_addr], 0
    //clear breakpoint
    bphwc   bp_NtQueryInformationProcess
```

Debugger Detection > Debugger Interrupts

- INT1 and INT3 does not invoke the exception handler (by default) if process is debugged since they are typically handled by the debugger
- If after INT1/INT3 the exception handler is not invoked, it means process is being debugged
- Flags can be set inside the exception handler to mark that it had been executed
- Some packers use kernel32!DebugBreak() since it invokes INT3

Debugger Detection > Debugger Interrupts

- What is a CONTEXT:
  - Contains the current state of the thread
  - Retrieved via GetThreadContext()
  - Also passed to the exception handler via ContextRecord parameter (esp+0xc), contains the state of the thread when the exception occurred

```
lkd> dt _CONTEXT
   +0x000 ContextFlags     : Uint4B
   +0x004 Dr0              : Uint4B
   :::
   +0x018 Dr7              : Uint4B
   :::
   +0x08c SegGs            : Uint4B
   +0x090 SegFs            : Uint4B
   :::
   +0x0b0 Eax              : Uint4B
   +0x0b4 Ebp              : Uint4B
   +0x0b8 Eip              : Uint4B
```

Debugger Detection > Debugger Interrupts

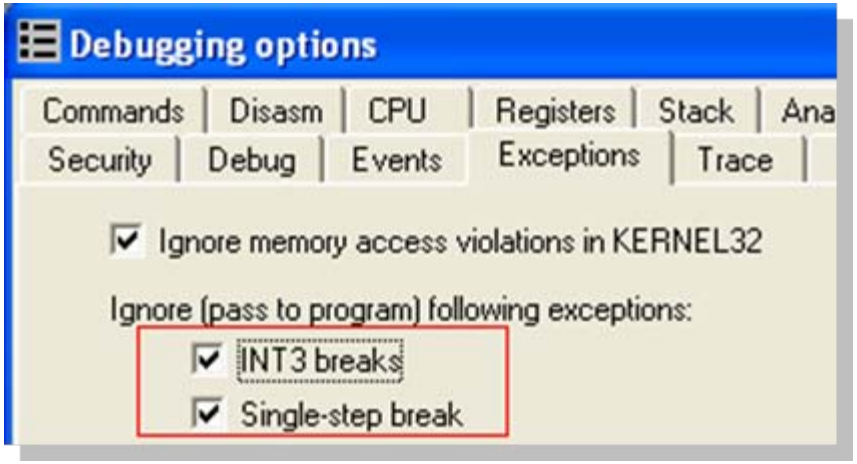
- Example: Set a Flag (EAX) in the exception handler

```
;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
```

Debugger Detection > Debugger Interrupts

- Solution: In OllyDbg, allow debugger interrupts to be passed to the exception handler via Shift+F7/F8/F9
- The exception handler address can be located via View->SEH Chain
- Another solution is to automatically pass debugger interrupts/exceptions to the exception handler via configuration: Debugging Options->Exceptions



Debugger Detection > Timing Checks

- Several CPU cycles are spent by debugger event handing code, reverser stepping thru instructions (and thinking)
- Packers check the time spent between instructions, if time spent passed a specific threshold, process is probably being debugged
- Packers use the following for time measurements:
  - RDTSC instruction (Read Time-Stamp Counter)
  - kernel32!GetTickCount()
  - TickCountLow and TickCountMultiplier in SharedUserData

Debugger Detection > Timing Checks

- Example: Using RDTSC to check time spent

```
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
```

Debugger Detection > Timing Checks

- Solutions:
  - Avoid stepping thru unimportant code containing timing checks, just set a breakpoint and perform a run
  - Set a breakpoint in GetTickCount()
  - Olly Advanced has a another solution against the RDTSC check:
    - Set Time Stamp Disable Bit (TSD) in CR4. Once set, if RDTSC is executed in privilege level != 0, a General Protection (GP) exception is triggered
    - Interrupt Descriptor Table (IDT) is set up to handle GP. If GP is because of an RDTSC instruction, increment the returned timestamp value from the previous call by 1
  - Note that the last solution may cause system instability

Debugger Detection > SeDebugPrivilege

- SeDebugPrivilege is disabled on a process access token by default
- OllyDbg/WinDbg enables the SeDebugPrivilege privilege in their access token
- The debugged process will inherit the access token of the debugger, including SeDebugPrivilege
- Note that SeDebugPrivilege is only granted for administrators by default
- Packers indirectly check if SeDebugPrivilege is enabled by attempting to open the CSRSS.EXE process - CSRSS.EXE is only accessible to SYSTEM, SeDebugPrivilege overrides the security descriptor

Debugger Detection > SeDebugPrivilege

- Example: Attempt 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
```

- Solution: Patch ntdll!NtOpenProcess() to return 0xC0000022 (STATUS_ACCESS_DENIED) if passed PID is for CSRSS.EXE

Debugger Detection > Parent Process

- Packers checks if parent process of the current process is not explorer.exe, if not, process is probably being debugged
- Implementation involves:
  - Retrieve the current process' PID via TEB.ClientId (fs:[20]) or via kernel32!GetCurrentProcessId()
  - Enumerate process:
    - Find PID of explorer.exe
    - Find Parent process PID of current process
  - Check if Parent Process PID != PID of explorer.exe
- Solution: kernel32!Process32NextW() can be patched to always return 0x0 – packers may choose to skip the check

Debugger Detection > DebugObject

- Involves checking if a debugging session is active by checking if the number of objects of type DebugObject is not 0
- A DebugObject is created for every debugging session
- DebugObject can be queried via ntdll!NtQueryObject(ObjectAllTypeInformation)
  - Returns the following structure:

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

  - OBJECT_TYPE_INFORMATION structure:

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

Debugger Detection > DebugObject

- Solutions:
  - Returned OBJECT_ALL_INFORMATION.NumberOfObjectTypes can be manipulated to 0
  - Olly Advanced injects code into NtQueryObject to zero out the entire returned OBJECT_ALL_INFORMATION buffer

Debugger Detection > Debugger Window

- Packers also identify if a debugger is running by checking for existence of debugger Windows
- Debugger windows has predefined class names:
  - OLLYDBG for OllyDbg
  - WinDbgFrameClass for WinDbg
- Use FindWindow() / FindWindowEx() to check for existence of debugger windows

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

.szWindowClassOllyDbg   db "OLLYDBG",0
```

- Solution: Set a breakpoint on FindWindow(), then, manipulate lpClassName param or return value

Debugger Detection > Debugger Process

- Packers also identify if a debugger is active via debugger process
- Just involves enumerating all process and check if PROCESSENTRY32.szExeFile is a debugger EXE name:
  - OLLYDBG.EXE
  - windbg.exe, etc.
- Some packers reads the process memory and look for debugger-related strings (eg: “OLLYDBG”)
- Solution: Patch kernel32!Process32NextW() to always fail to prevent process enumeration

Debugger Detection > Device Drivers

- Classic technique for detecting kernel mode debuggers
- Fairly simple, involves invoking kernel32!CreateFileA() against well-known device names used by kernel mode debuggers
- Technique is also used to detect existence of system monitors (FileMon, RegMon)
- Solution: Set a breakpoint in kernel32!CreateFileW(), then manipulate the return value to INVALID_HANDLE_VALUE

```
    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
```

Debugger Detection > OllyDbg: Guard Pages

- Specific to OllyDbg
- OllyDbg has a on-access/write memory breakpoint feature which is separate from hardware breakpoints
- Feature is implemented via Guard Pages
- Guard Pages provides a way for applications to be notified if a memory access/write on specific pages occurred
- Guard Pages are set via PAGE_GUARD page protection modifier and triggers STATUS_GUARD_PAGE_VIOLATION (0x80000001) exception if accessed
- If process is being debugged in OllyDbg, the exception handler will not be called

Debugger Detection > OllyDbg: Guard Pages

- Example: Setting up and triggering a STATUS_GUARD_PAGE VIOLATION

```
; set up exception handler
:::
; 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
mov     byte [eax],0xC3
; then set the PAGE_GUARD attrib of page
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 STATUS_GUARD_PAGE_VIOLATION
    ; exception
    call    [.pAllocatedMem]
    ; check if marker had not been changed
    test    eax,eax
    je      .debugger_found
    :::
.exception_handler
    ;EAX = CONTEXT record
    mov     eax,[esp+0xc]
    ;set marker (CONTEXT.EAX) to
    mov     dword [eax+0xb0],0xffffffff
    xor     eax,eax
    retn
```

Debugger Detection > OllyDbg: Guard Pages

- Solution: Deliberately trigger an exception so that the exception handler will be called
  - In the last example, perform a INT3, then a RETN
- If the exception handler checks the exception code, reverser needs to set a breakpoint in the exception handler, then change ExceptionRecord.ExceptionCode to STATUS_PAGE_GUARD_VIOLATION manually

IBM Global Services

The Art Of Unpacking

Breakpoint and Patching Detection

IBM Internet Security Systems

Ahead of the threat.™

Breakpoint/Patching Detection > Software Breakpoints

- Software breakpoints are set by replacing the instruction at the target address by 0xCC (INT3 / Breakpoint interrupt)
- Packers identify software breakpoints by searching for 0xCC in the unpacking stub or APIs
- Some packers apply operation on the compared byte value so the check is not obvious:

```
if(byte XOR 0x55 == 0x99) then breakpoint found
Where: 0x99 == 0xCC XOR 0x55
```

- Solution:
  - Use hardware breakpoints
  - Set a breakpoint in UNICODE versions of the API (LoadLibraryExW instead of LoadLibraryA) or Native APIs

Breakpoint/Patching Detection > Hardware Breakpoints

- Hardware breakpoints are set via Debug Registers
- Debug Registers:
  - Dr0 – Dr3: Address of breakpoints
  - Dr6 – Debug Status: Determine what breakpoint had been triggered
  - Dr7 – Debug Control: Flags to control the breakpoints such as break on-read or on-write, etc.
- Debug registers are not accessible in Ring 3
- Debug registers are checked via the CONTEXT record passed to the exception handler

Breakpoint/Patching Detection > Hardware Breakpoints

- Example: Setup exception handler and check Context.Dr*x*

```
; set up exception handler
:::
; initialize marker
xor     eax,eax
; throw an exception
mov     dword [eax],0
; restore exception handler
:::
; 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 are 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
    :::
```

Breakpoint/Patching Detection > Patching Detection

- Identifies if part of the unpacking stub had been modified (eg: checks are disabled by the reverser)
- Detects software breakpoints as a side effect
- Involves performing a checksum on a specific range of code/data
- Some use simple checksums, while other use intricate checksum/hash algorithms
- Solution:
  - Avoid setting software breakpoints if checksum routines exists
  - On patched code, try setting an on-access breakpoint on the modified code, once the breakpoint is hit, analyze the checksum code and change the resulting checksum to the correct value

IBM Global Services

The Art Of Unpacking
Anti-Analysis

IBM Internet Security Systems
Ahead of the threat.™

Anti-Analysis > Encryption and Compression

- Encryption: Packers usually encrypt both the unpacking stub and the protected executable
- Algorithms to encrypt ranges from very simple XOR loops to complex loops the perform several computations
- Decryption loops are easy to recognize: fetch -> compute -> store operation
- Encryption algorithms and unpacking stub varies on polymorphic packers

Anti-Analysis > Encryption and Compression

- Example: Polymorphic packer decryption loop (register swapping, garbage codes)

```
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
```

```
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
```

Anti-Analysis > Encryption and Compression

- Compression: Reduce the size of the protected executable
- Obfuscation side effect because both the protected executable code and data became compressed data
- Examples:
  - UPX: NRV (Not Really Vanished), LZMA
  - FSG: aPLib
  - Upack: LZMA
- Solution: Determine how the decryption/decompression loop ends and set a breakpoint
- Remember, breakpoint detection code may exist on the decryption/decompression loop

Anti-Analysis > Garbage Code and Permutation

- Garbage code: Garbage codes are effective way to confuse a reverser
- They hide the real purpose of the code
- Adds effectiveness to other anti-reversing techniques by hiding them
- Effective garbage code are those that look like legitimate/working code

Anti-Analysis > Garbage Code and Permutation

- Another Example
  - Garbage operations
  - JMPs

```
0044A225  MOV ESI,DWORD PTR SS:[ESP]
0044A23F  ADD DWORD PTR DS:[ESI],3CB3AA25
0044A268  SUB ESI,-4
0044A280  XOR DWORD PTR DS:[ESI],33B568E3
0044A29D  MOV EAX,4
0044A2A2  ADD ESI,EAX
0044A2B0  NOT DWORD PTR DS:[ESI]
```

```
0044A21A  JMP SHORT sample.0044A21F
0044A21C  XOR DWORD PTR SS:[EBP],6E4858D
0044A223  INT 23
0044A225  MOV ESI,DWORD PTR SS:[ESP]
0044A228  MOV EBX,2C322FF0
0044A22D  LEA EAX,DWORD PTR SS:[EBP+6EE5B321]
0044A233  LEA ECX,DWORD PTR DS:[ESI+543D583E]
0044A239  ADD EBP,742C0F15
0044A23F  ADD DWORD PTR DS:[ESI],3CB3AA25
0044A245  XOR EDI,7DAC77F3
0044A24B  CMP EAX,ECX
0044A24D  MOV EAX,5ACAC514
0044A252  JMP SHORT sample.0044A257
0044A254  XOR DWORD PTR SS:[EBP],AAE47425
0044A25B  PUSH ES
0044A25C  ADD EBP,5BAC5C22
0044A262  ADC ECX,3D71198C
0044A268  SUB ESI,-4
::: more garbage code:::
0044A280  XOR DWORD PTR DS:[ESI],33B568E3
0044A286  LEA EBX,DWORD PTR DS:[EDI+57DEFEE2]
0044A28C  DEC EDI
0044A28D  SUB EBX,7ECDAE21
0044A293  MOV EDI,185C5C6C
0044A298  MOV EAX,4713E635
0044A29D  MOV EAX,4
0044A2A2  ADD ESI,EAX
0044A2A4  MOV ECX,1010272F
0044A2A9  MOV ECX,7A49B614
0044A2AE  CMP EAX,ECX
0044A2B0  NOT DWORD PTR DS:[ESI]
```

Anti-Analysis > Garbage Code and Permutation

- Code Permutation: Simple instructions are translated into more complex series of instructions
- Used by more advanced packers – requires understanding of the instructions
- Simple illustration:

```
mov     eax,ebx
test    eax,eax
```

```
push    ebx
pop     eax
or      eax,eax
```

Anti-Analysis > Garbage Code and Permutation

- Example: Code permutation

```
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
```

```
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
```

Anti-Analysis > Garbage Code and Permutation

- Solutions:
  - Identify if the instructions between the garbage codes are worth understanding
  - Try using “trace markers” by setting breakpoints on mostly used APIs by packers (eg:VirtualAlloc, LoadLibrary, GetProcAddress, etc.). If something went wrong between trace markers, then, it is time to perform a detailed trace
  - OllyDbg + VMWare is useful to save trace state so the reverser can go back to a specific state
  - On-memory access/write breakpoints on interesting code/data are also useful

Anti-Analysis > Anti-Disassembly

- Obfuscate the disassembly produced by disassemblers/debugger
- One method involves:
  - Inserting a garbage byte
  - Add a conditional branch to the garbage byte
  - The condition for the conditional branch will always be FALSE
- The disassembler will follow and disassemble the garbage byte and produce an incorrect output

Anti-Analysis > Anti-Disassembly

- Example:
  - Debugger Detection via PEB.BeingDebugged flag

```
mov     eax,dword [fs:0x18]
mov     eax,dword [eax+0x30]
movzx   eax,byte [eax+0x02]
test    eax,eax
jnz     .debugger_found
```

```
;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
```

Anti-Analysis > Anti-Disassembly

- Example: WinDbg and OllyDbg Disasm Output

```
0040194a  push    0x401954
0040194f  stc
00401950  jnb     image00400000+0x1953 (00401953)
00401952  ret
00401953  jmp     dword ptr [ecx+0x18]
00401957  add     [eax],al
00401959  add     [eax+0x64],ch
0040195c  sbb     [eax],eax
0040195f  clc
00401960  jb      image00400000+0x
00401962  ret
00401963  dec     dword ptr [ebx+0
00401969  inc     eax
0040196a  add     al,[ebp+0x310775
```

```
0040194A  PUSH 00401954
0040194F  STC
00401950  JNB SHORT 00401953
00401952  RETN
00401953  JMP DWORD PTR DS:[ECX+18]
00401957  ADD BYTE PTR DS:[EAX],AL
00401959  ADD BYTE PTR DS:[EAX+64],CH
0040195C  SBB DWORD PTR DS:[EAX],EAX
0040195F  CLC
00401960  JB SHORT 00401963
00401962  RETN
00401963  DEC DWORD PTR DS:[EBX+B60F3040]
00401969  INC EAX
0040196A  ADD AL,BYTE PTR SS:[EBP+310775C0]
```

Anti-Analysis > Anti-Disassembly

- Example cont.: IDAPro Disassembly Output

```
0040194A            push    (offset loc_401953+1)
0040194F            stc
00401950            jnb     short loc_401953
00401952            retn
00401953 ; ---------------------------------------------------------------
00401953
00401953 loc_401953:                            ; CODE XREF: sub_401946+A
00401953                                        ; DATA XREF: sub_401946+4
00401953            jmp     dword ptr [ecx+18h]
00401953 sub_401946 endp
00401953
00401953 ; ---------------------------------------------------------------
00401957            db    0
00401958            db    0
00401959            db    0
0040195A            db  68h ; h
0040195B            dd offset unk_401964
0040195F            db 0F8h ; ·
00401960            db  72h ; r
00401961            db    1
00401962            db 0C3h ; +
00401963            db 0FFh
00401964 unk_401964 db  8Bh ; ï             ; DATA XREF: text:0040195B
::::
```

IBM Global Services

The Art Of Unpacking
Debugger Attacks

IBM Internet Security Systems
Ahead of the threat.™

Debugger Attacks > Misdirection / Stopping via Exceptions

- Packers employ several techniques so that tracing is not linear, and so that the code is not easily understandable
- One common technique is by throwing caught exceptions
- The exception handler will set the next EIP
- Packers also uses exceptions to pause execution if process is being debugged

Debugger Attacks > Misdirection / Stopping via Exceptions

- Example: Misdirection via Exception

```
    ; 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
```

Debugger Attacks > Misdirection / Stopping via Exceptions

- Solution: If the exception is only for transferring execution to different parts of the code, exceptions can be automatically passed to exception handler

- The reverser can set a breakpoint on the exception handler, then press Shift+ F7/F8/F9

Debugger Attacks > Blocking Input

- Prevent the reverser from controlling the debugger
- User32!BlockInput() block keyboard/mouse inputs
- Can be effective if hidden by garbage codes
- Can baffle the reverser if not identified
- Example:

```
push    TRUE
call    [BlockInput]    ;Block input
; ...Unpacking code...
push    FALSE
call    [BlockInput]    ;Unblock input
```

- Solution: Patch user32!BlockInput() to just perform a RETN
- Pressing CTRL+ALT+DELETE to manually unblock input

Debugger Attacks > ThreadHideFromDebugger

- Prevents debugging events from reaching the debugger
- Can be set by ntdll!NtSetInformationThread(ThreadHideFromDebugger)
- Internally, it sets the HideThreadFromDebugger field of the ETHREAD kernel structure
- Example:

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

- Solution: Set a breakpoint on NtSetInformationThread(), and then prevent the call from reaching the kernel.

Debugger Attacks > Disabling Breakpoints

- Some packers also disable breakpoints
- Hardware breakpoints are disabled via the CONTEXT record passed to exception handlers
- Software breakpoints can also be disabled by replacing identified 0xCC (INT3s) with a random/predefined byte, thus, also causing a corruption

Debugger Attacks > Disabling Breakpoints

- Example: Clearing Dr0-Dr7 via ContextRecord

```
    ; 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
```

Debugger Attacks > Disabling Breakpoints

- Solution: Clearly, if hardware breakpoints are detected, use software breakpoints, vice versa
- Also try using on access/write memory breakpoint feature of OllyDbg
- Try setting software breakpoints inside UNICODE versions or native APIs since they are not being checked by some packers

Debugger Attacks > Unhandled Exception Filter

- MSDN: If an exception reached the unhandled exception filter and that the process is being debugged, the registered top level exception filter will not be called
- kernel32!SetUnhandledExceptionFilter() sets the top level exception filter
- Some packers manually set the exception filter by setting kernel32!_BasepCurrentTopLevelFilter
- Solution: Similar to the solution to the DebugPort debugger detection technique – manipulate return value of ntdll!NtQueryInformationProcess()
  - UnhandledExceptionFilter calls NtQueryInformationProcess (ProcessDebugPort) to determine if process is being debugged

Debugger Attacks > Unhandled Exception Filter

- Example: Throw an exception and set Context.EIP on exception filter

```
;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
```

Debugger Attacks > OllyDbg: OutputDebugString() Format String Bug

- Specific to OllyDbg
- OllyDbg is known to be vulnerable to a format string bug which can cause it to crash or execute arbitrary code
- Triggered by an improper string parameter passed to kernel32!OutputDebugString()
- Example:

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

- Solution: Patch OutputDebugString() to just perform a RETN

IBM Global Services

The Art Of Unpacking

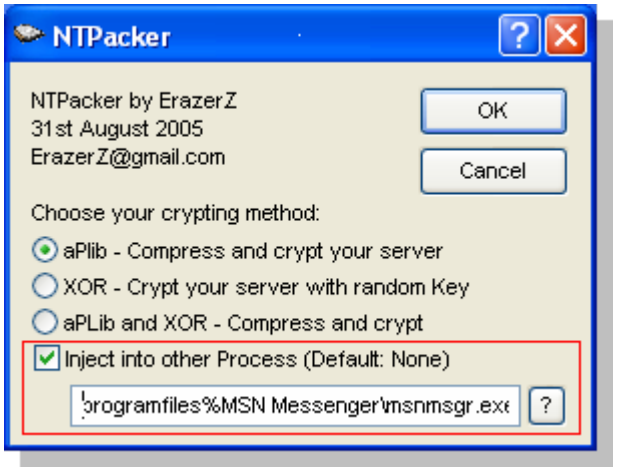
Advanced and Other Techniques

IBM Internet Security Systems

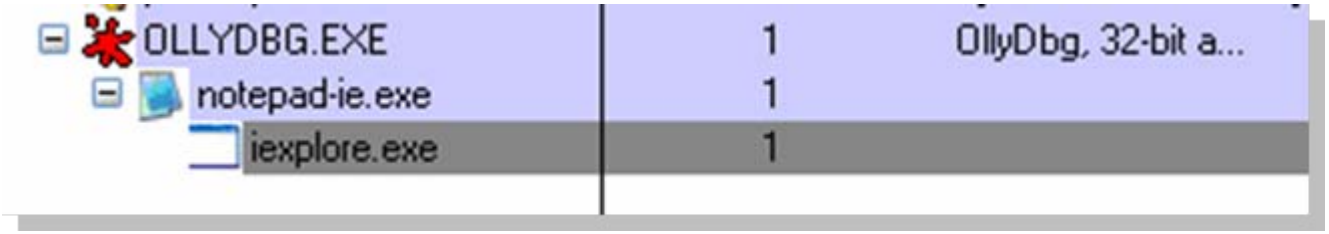
Ahead of the threat.™

Advanced / Other Techniques > Process Injection

- Process injection became a feature of some packers



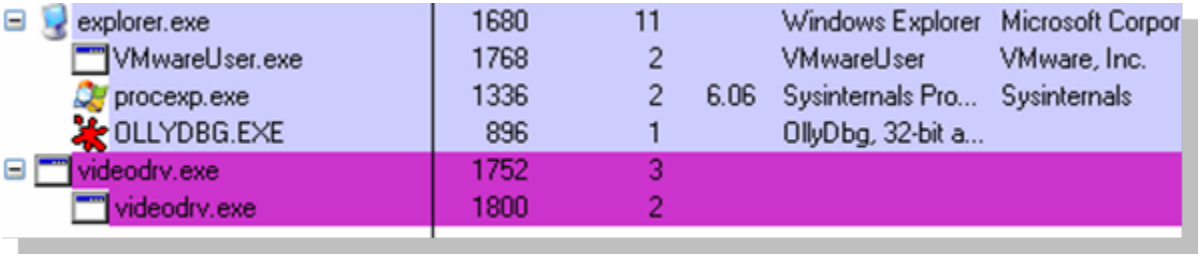
- Involves selecting a host process (eg: itself, explorer.exe, iexplore.exe), then injecting code into the host process



- A method to bypass some firewalls

Advanced / Other Techniques > Debugger Blocker

- Introduced by the Armadillo packer
- Prevents a debugger from attaching to a protected process
- Method involves a spawning and debugging a protected process



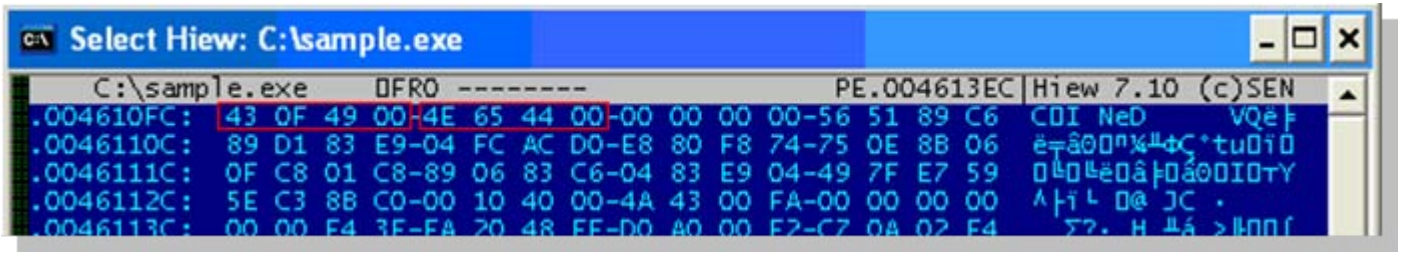
explorer.exe	1680	11		Windows Explorer	Microsoft Corpor
VMwareUser.exe	1768	2		VMwareUser	VMware, Inc.
procexp.exe	1336	2	6.06	Sysinternals Pro...	Sysinternals
OLLYDBG.EXE	896	1		OllyDbg, 32-bit a...	
videodrv.exe	1752	3			
videodrv.exe	1800	2			

- Since the protected process is already being debugged, another debugger can't attach to the process

Advanced / Other Techniques > TLS Callbacks

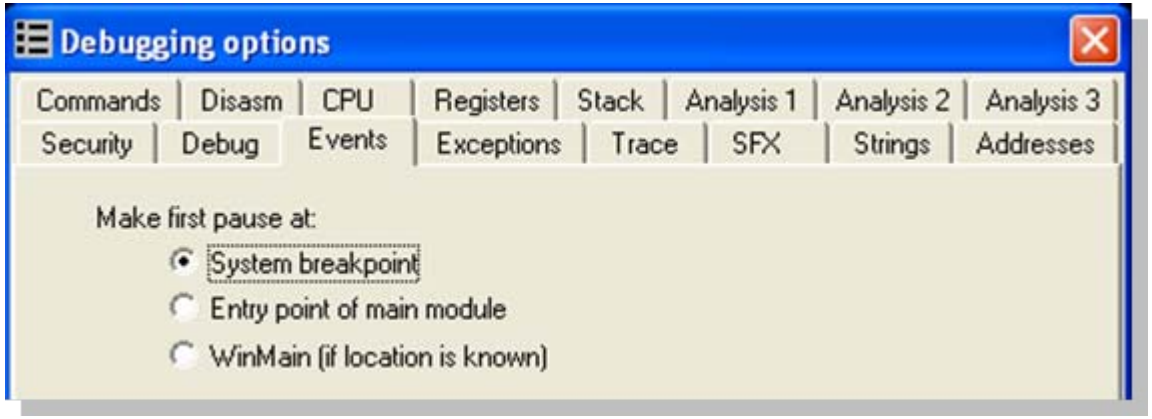
- A technique for code to execute before the actual entry point
- TLS callbacks can be identified by PE file parsing tools (eg: pedump)

```
TLS directory:
  StartAddressOfRawData: 00000000
  EndAddressOfRawData:   00000000
  AddressOfIndex:        004610F8
  AddressOfCallBacks:    004610FC
  SizeOfZeroFill:        00000000
  Characteristics:       00000000
```



Advanced / Other Techniques > TLS Callbacks

- TLS callbacks can be traced by breaking inside ntdll!_LdrpInitializeProcess (system breakpoint) just before TLS callbacks are called:



Advanced / Other Techniques > Stolen Bytes

- Prevent complete reconstruction via process dumping
- Portions of the code (usually entry point) is removed (stolen) by the packer and executed from an allocated memory

```
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
```

```
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
```

Advanced / Other Techniques > API Redirection

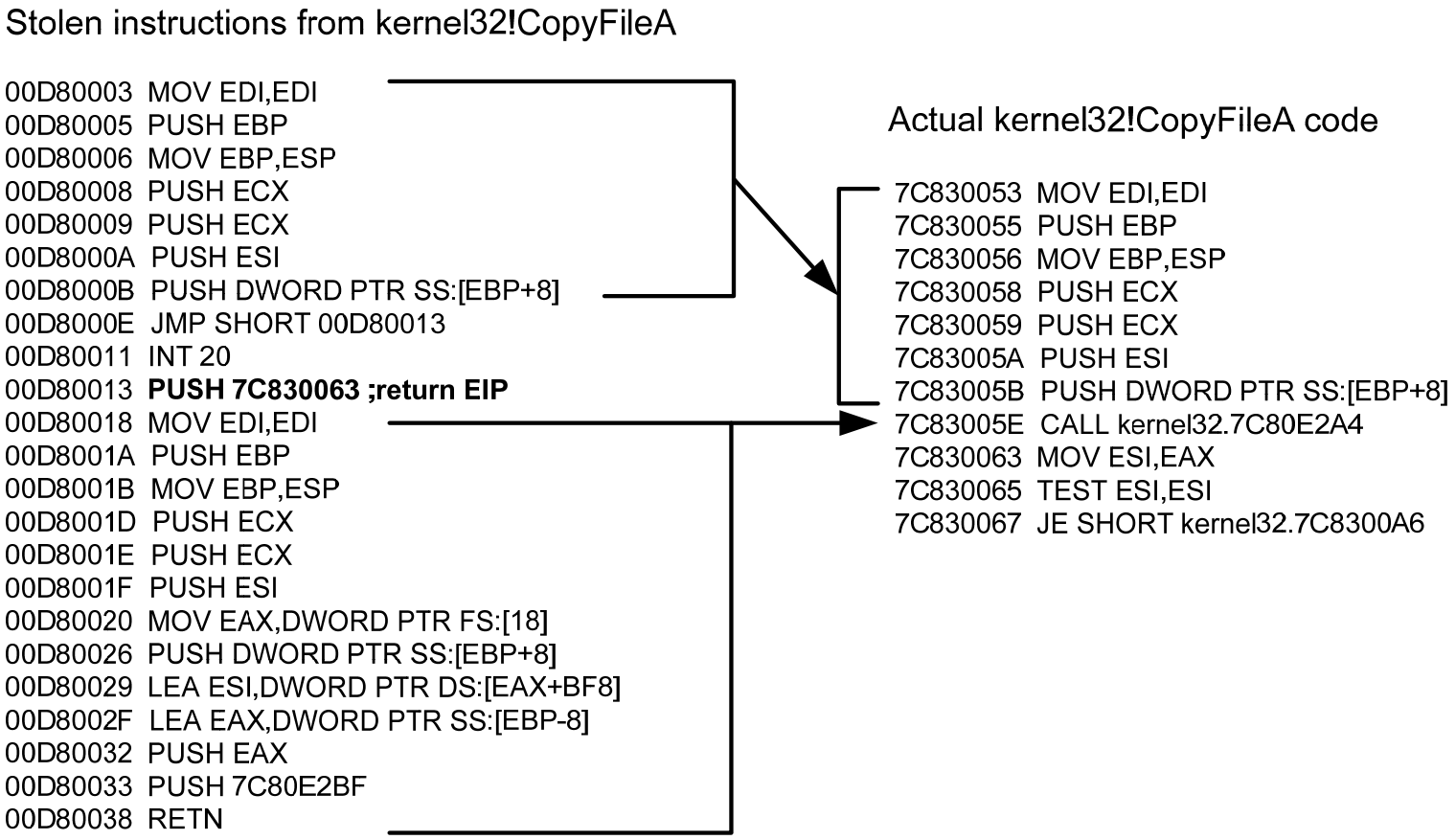
- Prevents import table rebuilding
- API calls are redirected to code in allocated memory
- Parts of the API code are also copied and executed from an allocated memory, then control is transferred in the middle of the API code in the DLL image
- Example: Redirected kernel32!CopyFileA()

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

```
004056B8  CALL 00D90000
```

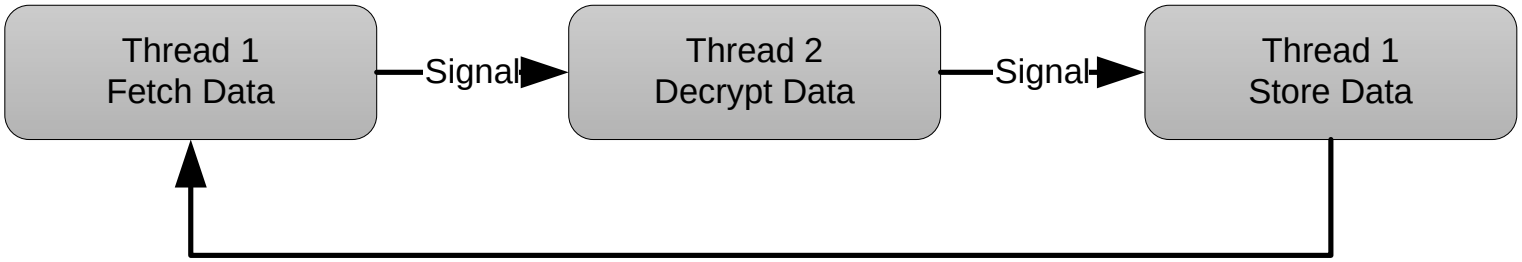
Advanced / Other Techniques > API Redirection

- Example Cont.: Illustration of the redirected kernel32!CopyFileA() API



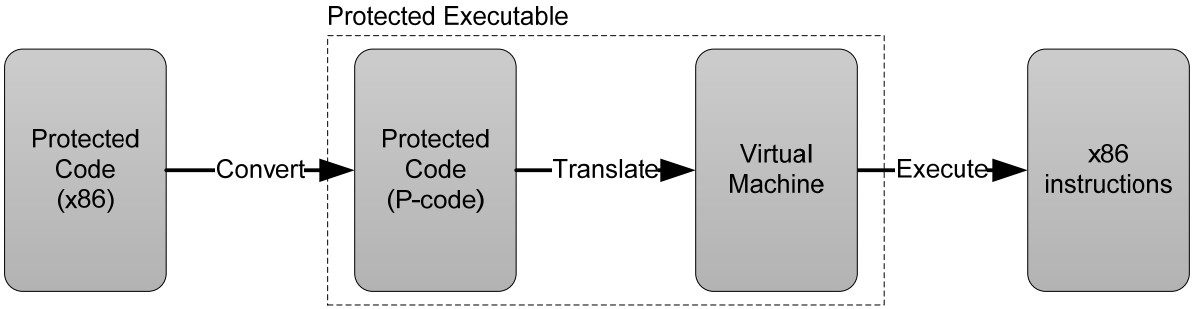
Advanced / Other Techniques > Multi-Threaded Packers

- Complicates tracing and the difficulty of understanding the code increases
- Example: PECrypt uses a second thread to perform decryption of a data fetched by the main thread



Advanced / Other Techniques > Virtual Machines

- Eventually, the protected code needs to be decrypted and executed in memory leaving it vulnerable to process dumping and static analysis
- Modern packers solves this by transforming the protected code into p-codes and executing them in virtual machines
- Illustration:



- This makes reversing more time consuming since this requires reversing the p-code structure and translation

IBM Global Services

The Art Of Unpacking

Tools

IBM Internet Security Systems

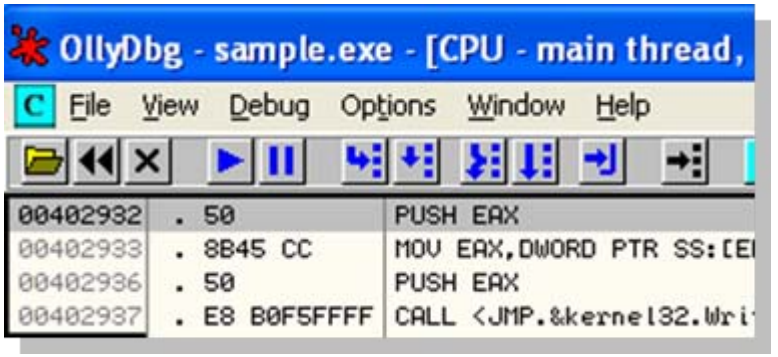
Ahead of the threat.™

Tools > OllyDbg, OllyScript, Olly Advanced

- OllyDbg

  http://www.ollydbg.de/

  – Powerful Ring 3 debugger.

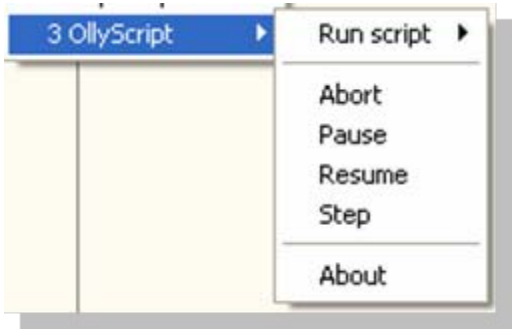


- OllyScript

  http://www.openrce.org/downloads/details/106/OllyScript

  – Allows automation of setting/handling breakpoints

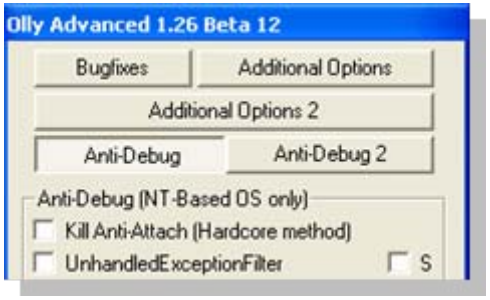
  – Useful in performing repetitive tasks



- Olly Advanced

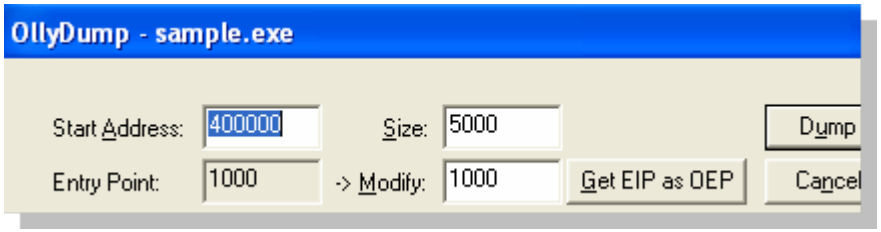
  http://www.openrce.org/downloads/details/241/Olly_Advanced

  – An armor to Ollydbg against anti-debugging and much more…

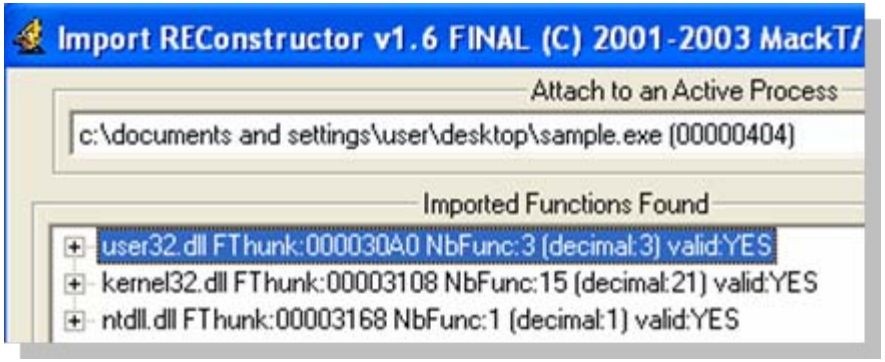


Tools > OllyDump and ImpRec

- OllyDump
  http://www.openrce.org/downloads/details/108/OllyDump
  - OllyDbg plugin for process dumping and import table rebuilding



- ImpRec
  http://www.woodmann.com/crackz/Unpackers/Imprec16.zip
  - Stand-alone tool for process dumping and excellent import table rebuilding capability



Thank you!

Questions?

Mark Vincent Yason
Malcode Analyst
X-Force Research & Development
myason@us.ibm.com