Anti-debugging techniques

Malware Analysis Seminar

Meeting 3

Cody Cutler, Anton Burtsev

Debugger detection

PEB.BeingDebuggedFlag

- BeingDebugged flag in PEB
  - Call kernel32!IsDebuggerPresent()

```
call   [IsDebuggerPresent]
test   eax,eax
jnz    .debugger_found
```

  - Check PEB.BeingDebugged directly

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

Solution

- Patch PEB.BeingDebugged with 0x0
  - OllyDbg data window (Ctrl+G) type fs:[30]
  - OllyDbg advanced plugin has an option to set BeingDebugged to 0x0.

PEB.NtGlobalFlag, Heap Flags

- Additional flags are set for a debugged process
  - PEB.NtGlobalFlag (offset 0x68, normal 0x0)
    - FLG_HEAP_ENABLE_TAIL_CHECK (0x10)
    - FLG_HEAP_ENABLE_FREE_CHECK (0x20)
    - FLG_HEAP_VALIDATE_PARAMETERS (0x40)
  - PEB.HeapProcess.{Flags, ForceFlags}
    - HEAP_TAIL_CHECKING_ENABLED (0x20)
    - HEAP_FREE_CHECKING_ENABLED (0x40)

Example

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


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

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

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

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

Solution

- OllyDBG script

```
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
```

DebugPort: CheckRemoteDebuggerPresent()

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

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

DebugPort: NtQueryInformationProcess()

```
NTSTATUS NTAPI NtQueryInformationProcess(
    HANDLE ProcessHandle,
    PROCESSINFOCLASS ProcessInformationClass,
    PVOID ProcessInformation,
    ULONG ProcessInformationLength,
    PULONG ReturnLength
)

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
```

```
    var bp_NtQueryInformationProcess

    // 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

bp_handler_NtQueryInformationProcess_continue:
    run
```

Debugger interrupts

- Update magic values from inside INT3 and INT1 exception handlers
  - Values are not set if exceptions are handled by the debugger itself
- Example
  - Set EAX to 0xFFFFFFFF via CONTEXT record

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

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

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

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

    :::

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

Solution

- When stopped due to a debugger interrupt
  - Identify the exception handler address
    - via View->SEH Chain
  - Set a breakpoint on the exception handler
  - Shift+F9 – pass exception
- Alternative: OllyDBG automatic exception passing
  - Options->Debugging Options->Exceptions-> “Ignore following exceptions”
  - “INT 3 breaks”, “Single-step breaks”

Timing checks

- Use rdtsc to detect that some instructions take too long due to single-stepping

- Other sources of time:
  - kernel32!GetTickCount()
  - TickCountLow, and TickCountMultiplier fields of the SharedUserData structure (always located at 0xc)
  - Caches, branch predictors

Example

```
rdtsc
mov ecx,eax
mov ebx,edx

; ... some code

;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
```

Solution

- Avoid single-stepping
  - Set breakpoint after second rdtsc
- Set breakpoint, and patch sources of time
  - GetTickCount()
  - Disable rdtsc user-level access (OllyDBG)
    - Time Stamp Disable bit in CR4
    - OllyBDG handles General Protection exception
    - You can increment TSC value by 1

SeDebugPrivilege

- Debugged processes have SeDebugPrivilege
  - An indirect check by opening CSRSS.EXE

```
; 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, we're being debugged
test eax,eax
jnz .debugger_found
```

Solution

- Break and patch ntdll!NtOpenProcess()
  - If PID is equal to CSRSS.EXE
  - EAX to 0xC0000022 (STATUS_ACCESS_DENIED)

Parent process

- Typically explorer.exe is your parent
  - Retrieve PID via TEB.ClientId, or GetCurrentProcessId()
  - List all processes Process32First/Next()
- Solution (OllyAdvanced):
  - Always fail Process32Next() hoping it will fail the PID check
    - Patch Process32Next() to always return error code and exit

Debugger detection

- Number of kernel objects of type DebugObject
  - NtQueryObject()
  - OllyDBG script (see NtQueryInformationProcess())
    - zero-out size of returned array
    - zero-out array content
- Debugger window
  - user32!FindWindow, user32!FindWindowEx

Debugger detection (contd)

- Debugger process
  - List all processes and look for common debugger names
    - Process32First/Next()
  - Read process memory and look for known strings
    - kernel32!ReadProcessMemory()
- Device drivers: SoftICE, Regmon, Filemon
  - Open well-known device names
    - kernel32!CreateFile()

Guard pages

- Debuggers use page-level protection to implement watchpoints
- Page guards are not fully virtualized by debuggers
  - Allocate and guard a page
  - Put some code there (like RETN)
  - Jump to it
  - If debugger uses page guarding, exception will be suppressed
    - Magic values will not be updated
- You can do the same attack with any resource used by a debugger, and not properly virtualized

Solution

- Force the exception
  - If the page contains RETN instruction, replace it with INT3, RETN
  - Like above, pass INT3 exception with Shift+F6
  - Let the handler run and update the magic values
  - Then RETN will proceed as normal
- More work, if exception handler checks for the exception vector
  - Patch it manually

Breakpoint detection

Software breakpoint detection

- Software breakpoints insert 0xCC (INT3) to trigger a breakpoint interrupt
- Scan the code for 0xCC

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

- Obfuscate the check

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

Solution

- Use hardware breakpoints
- Set breakpoints deeper in the API

- Emulate reads from memory?

Hardware breakpoint detection

- Debug registers are not directly accessible in Ring3
  - However they are passed to the exception handler as part of the CONTEXT struct
  - Set up an exception handler
  - Check CONTEXT struct
  - Pass an error code via EAX in CONTEXT struct from the handler back to the code
- Some packers use debug registers as input to decryption algorithms

Solution

- Use alternative breakpoints
  - Software
  - Page guards
- Set breakpoints deeper in the API

- I'm not sure why debuggers don't virtualize CONTEXT struct properly

Patching detection

- Identify if code of a packer was changed as an attempt to
  - Disable anti-debugging features
  - Set software breakpoints
- Solution
  - Identify checksum routine with an on-access watchpoint
  - Patch the checksum routine

Anti-analysis

Encryption and compression

- Packers encrypt both the protector code and the protected executable
- Encryption algorithms vary greatly
  - Polymorphic algorithms generate different output
    - Sometimes make a known packer unrecognizable
-

Encryption and compression

- Decryption routines recognizable as loops
  - Fetch, compute, store
- Example
  - Several XORs on a DWORD value

```
.loop:
    LODS DWORD PTR DS:[ESI]
    XOR EAX,EBX
    SUB EAX,12338CC3
    ROL EAX,10
    XOR EAX,799F82D0
    STOS DWORD PTR ES:[EDI]
    INC EBX
    LOOPD SHORT .loop ;decryption loop
```

Polymorphic examples

```
.loop:
    MOV BH,BYTE PTR DS:[EAX]
    INC ESI
    ADD BH,0BD
    XOR BH,CL
    INC ESI
    DEC EDX
    MOV BYTE PTR DS:[EAX],BH
    CLC
    SHL EDI,CL
    ::: More garbage code
    INC EDX
    DEC EDX
    DEC EAX
    JMP SHORT .foo
.foo:
    DEC ECX
    JNZ .loop ;decryption loop
```

```
.loop:
    MOV CH,BYTE PTR DS:[EDI]
    ADD EDX,EBX
    XOR CH,AL
    XOR CH,0D9
    CLC
    MOV BYTE PTR DS:[EDI],CH
    XCHG AH,AH
    BTR EDX,EDX
    MOVSX EBX,CL
    ::: More garbage code
    SAR EDX,CL
    NOP
    DEC EDI
    DEC EAX
    JMP SHORT .foo
.foo:
    JNZ .loop ;decryption loop
```

Solution

- Bypass the compression loop
  - Find a point at which loop terminates
  - Insert a breakpoint after the loop
    - Be aware of a breakpoint detection code inside the loop

Garbage code and code permutaion

- Insert a bunch of confusing instructions
  - Hide the real purpose of the code
  - Might look like a meaningful code
- Translate simple instructions in a series of less obvious equivalent instructions

```
mov  eax, ebx          push ebx
test eax, eax          pop  eax
                       or   eax, eax
```

Solution

- Don't try to understand them completely
- Skip through
  - Set breakpoints on commonly-used API
    - VirtualAlloc, VirtualProtect, LoadLibrary, GetProcAddress
  - Use API tracing tool and backtrack
    - If something goes wrong (anti-debugging) then trace
  - Set on-access watchpoints
    - See what code/data is touched
- Use VMM snapshots with OllyDBG

Anti-disassembly

- Garbage bytes
  - Jump to a garbage byte
  - A conditional branch is always FALSE at runtime

Example

```
    push .jmp_real_01          ;Anti-disassembly sequence #1
    stc
    jnc  .jmp_fake_01
    retn
.jmp_fake_01:
    db 0xff
.jmp_real_01:
    mov eax,dword [fs:0x18]
    push .jmp_real_02          ;Anti-disassembly sequence #2
    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
```

OllyDBG Disassembly

```
0040194A 68 54194000   PUSH 00401954
0040194F F9            STC
00401950 73 01         JNB SHORT 00401953
00401952 C3            RETN
00401953 FF64A1 18     JMP DWORD PTR DS:[ECX+18]
00401957 0000          ADD BYTE PTR DS:[EAX],AL
00401959 0068 64       ADD BYTE PTR DS:[EAX+64],CH
0040195C 1940 00       SBB DWORD PTR DS:[EAX],EAX
0040195F F8            CLC
00401960 72 01         JB SHORT 00401963
00401962 C3            RETN
00401963 FF8B 40300FB6 DEC DWORD PTR DS:[EBX+B60F3040]
00401969 40            INC EAX
0040196A 0285 C0750731 ADD AL,BYTE PTR SS:[EBP+310775C0]
```

Debugger attacks

Misdirection via exceptions

- Make sure that code is non-linear
  - Throw exceptions
    - Structured exception handling (SEH)
    - Vectored exceptions
  - Modify EIP inside exception handler

Blocking input

- Prevent reverser from controlling the debugger
  - user32!BlockInput()
  - Solution: Patch user32!BlockInput()
- Block debugging events
  - ntdll!NtSetInformationThread()
  - Solution: patch

Other

- Disable breakpoints
  - Block hardware breakpoints by patching CONTEXT passed to the exception handler
- Mess with other exception mechanisms
  - kernel32!UnhandledExceptionFilter()
- Format string vulnerabilities
  - Exist in OllyDBG and can be exploited
    - OllyDBG crashes

Advanced

Process injection

- Spawn a host process (iexplore.exe) as a suspended child
- kernel32!CreateProcess() with CREATE_SUSPENDED
- Retrieve child's context (kernel32!GetThreadContext())
- Retrieve image address (PEB.ImageBase (PEB is in the CONTEXT))
- Unmap original image (ntdll!NtUnmapViewOfSection())
- Allocate new image (kernel32!VirtualAllocEx())
- Write child's memory (kernel32!WriteProcessMemory())
- Update child's context (kernel32!SetThreadContext())
- Unsuspend (kernel32!ResumeThread())

Solution

- Break at WriteProcessMemory()
- Patch child's code to do an endless loop on entry point
- When parent resumes, attach debugger to the child
  - Restore original instructions
  - Continue debugging

Debugger blocker

- Spawn a process which becomes a debugger for the packed code
  - Tricky to attach a debugger
    - kernel32!DebugActiveProcess() will fail
- Solution: detach debugger
  - Inject kernel32!DebugActiveProcessStop() into debugger's code
  - Attach to a debugger
  - Break on kernel32!WaitForDebugEvent()
  - Inject the code

Nanomites

- Use self-debugging to take branches [Armadillo]
  - Replace branch instructions with INT3
  - Attach a debugger
  - Resolve branch targets in the debugger process

TLS callbacks

- Execute code before entry point
  - Debugger detection, decryption
- Solution
  - Identify TLS callbacks via PE file parsing tools (pedump)
  - Alternatively configure OllyDBG to break on load
    - ntdll!_LdrpInitializeProcess()

On-demand decompression

- Use page guards to decompress only accessed code [Shrinker]
  - Code size optimization
    - EXCEPTION_GUARD_PAGE
    - Hook ntdll!KiUserExceptionDispatcher()
- On-demand decryption [Armadillo]
  - Needs self-debugging

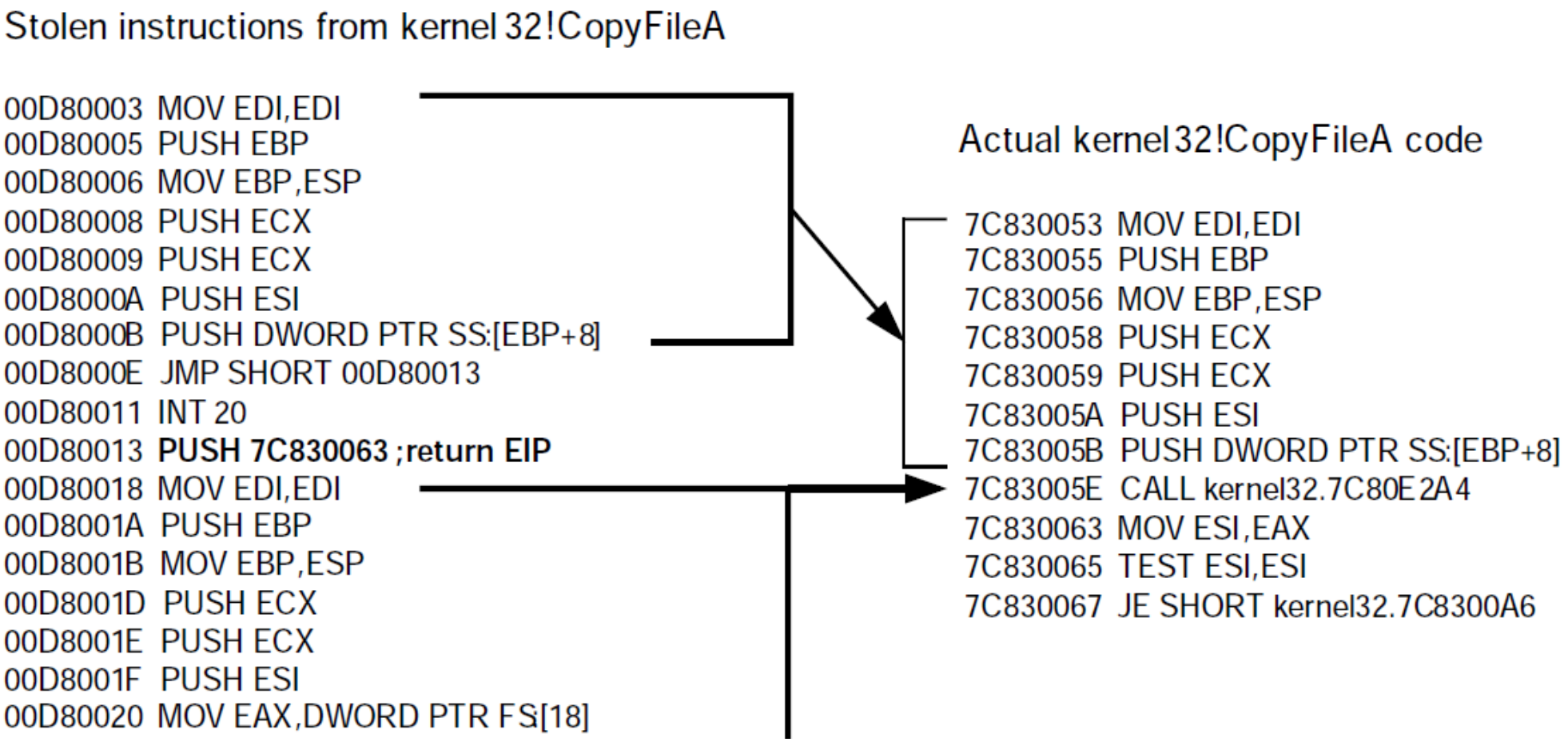
Stolen bytes

- Parts of the code are removed and executed from dynamically allocated memory [ASProtect]
  - Harder to dump executable

API redirection

- Import table is destroyed
- Calls are performed from stubs
  - Allocated dynamically
  - Obfuscate with stolen bytes
  - Sometimes entire copies of DLLs are loaded and used for API calls
    - Hard to set breakpoints on API functions

Example


Stolen instructions from kernel32!CopyFileA
00D80003 MOV EDI,EDI
00D80005 PUSH EBP
00D80006 MOV EBP,ESP
00D80008 PUSH ECX
00D80009 PUSH ECX
00D8000A PUSH ESI
00D8000B PUSH DWORD PTR SS:[EBP+8]
00D8000E JMP SHORT 00D80013
00D80011 INT 20
00D80013 PUSH 7C830063 ;return EIP
00D80018 MOV EDI,EDI
00D8001A PUSH EBP
00D8001B MOV EBP,ESP
00D8001D PUSH ECX
00D8001E PUSH ECX
00D8001F PUSH ESI
00D80020 MOV EAX,DWORD PTR FS:[18]
Actual kernel32!CopyFileA code
7C830053 MOV EDI,EDI
7C830055 PUSH EBP
7C830056 MOV EBP,ESP
7C830058 PUSH ECX
7C830059 PUSH ECX
7C83005A PUSH ESI
7C83005B PUSH DWORD PTR SS:[EBP+8]
7C83005E CALL kernel32.7C80E2A4
7C830063 MOV ESI,EAX
7C830065 TEST ESI,ESI
7C830067 JE SHORT kernel32.7C8300A6

Multi-threaded packers

- Another thread handles decryption or some other functionality [PECrypt]
  - Synchronized with the main thread
- Hard to understand

Virtual machines

- Translate packed code (p-code) on the fly [CodeVirtualizer, StarForce, VMProtect]
  - Ultimate anti-debugging technique
    - At no point in time code is directly visible in memory
  - p-code uses same techniques
    - polymorphism [Themida]
    - debugger detection [HyperUnpackMe2]
    - interpreter obfuscation [Themida, Virtual CPU]
- Solution
  - Implement new language disassembler
  - Hard but people do that

Acknowledgements

- This slides are mostly based on a BlackHat USA'07 presentation by Mark Vincent Yason
  - The Art of Unpacking. http://www.blackhat.com/presentations/bh-usa-07/Yason/Whitepaper/bh-usa-07-yason-WP.pdf