GPU Malwares – Now the evil hides in video cards

Alexandre Borges

Malwares – concepts and types and definitions

What's a malware?

- Any software that causes damage, data loss, gather information and provide external control of the system

- Pain, despair, anguish and suffering

Malwares Types

- Virus
- Worms
- Ransomware
- Rootkits
- Keyloggers
- Backdoor
- Botnet
- Trojans

Malwares – detection

- How do you know if your system is infected?
- Antivirus / IDS / Data Loss Prevention
- IPS/Firewall
- Sysinternals
- Memory Analysis (Volatility)

Analysis Techniques

Static Analysis

- Examine the code without running its instructions
- Reversing the code by using a disassembler (IDA Pro)

Dynamic Analysis

- Running the malware and observing its behavior.
- Using debuggers (Immunity, OllyDbg and WinDbg) to exame register, stack and memory.

Creating a Lab

- Virtual Machines without access to internal and external network.
- Operating systems used for running malwares: WinXP(32-bits) and Win7(64-bits)
- Kali Linux/Ubuntu to analyze the network traffic
- Snapshosts are useful
- Danger: Malwares can escape from a VM
- Sandboxes (Cuckoo, CFI Sandbox, and so on)

Basic Static Analysis – concepts, tools and methods

- Strings (strings.exe –a –n 6 <file>)
- Packed malwares → PEiD and plugins
- Portable Executable (PE) format → Import/Export Functions
- PEVIEW /pestudio / CFF Explorer
- Many critical functions and DLLs

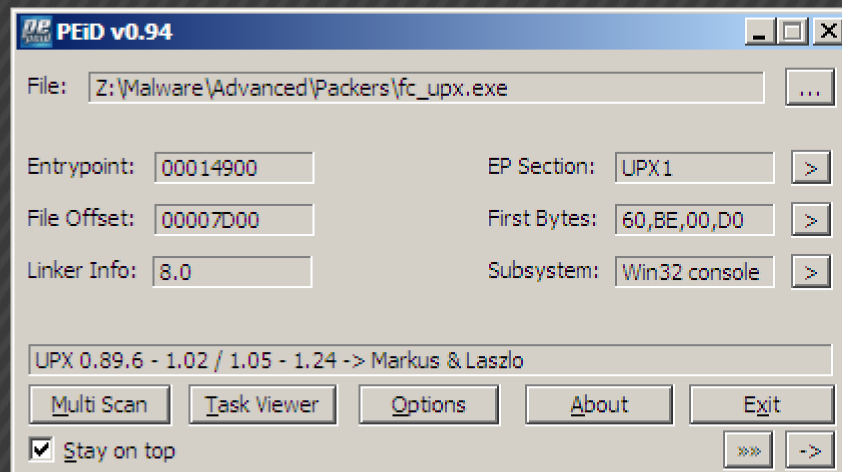
DLLs - examples

- Advapi32.dll → registry and service management
- WSock32.dll and Ws2_32.dll → network access
- Wininet.dll → HTTP/FTP protocols

Functions - examples

- CreateFileW
- WriteFile / ReadFile
- GetCurrentProcess
- SetWindowsHookEx
- RegistryClassExW
- RegSetValueExW
- WSAStartup (for allocating resources for network libs)
- InternetOpen/InternetOpenUrl
- InternetReadFile

PEiD



PEView

PEview - Z:\Malware\Advanced\Kickoff\syscheck.exe

File View Go Help

- syscheck.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .rdata
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - SECTION .text
  - SECTION .rdata
    - IMPORT Address Table
    - IMAGE_LOAD_CONFIG_DIRECTORY
    - IMPORT Directory Table
    - IMPORT Name Table
    - IMPORT Hints/Names & DLL Names
  - SECTION .data
  - SECTION .rsrc

pFile	Data	Description	Value
00000C00	00002278	Hint/Name RVA	01A0 GetProcAddress
00000C04	0000228A	Hint/Name RVA	0252 LoadLibraryA
00000C08	0000251C	Hint/Name RVA	0143 GetCurrentProcessId
00000C0C	00002506	Hint/Name RVA	0146 GetCurrentThreadId
00000C10	000024F6	Hint/Name RVA	01DF GetTickCount
00000C14	000024DC	Hint/Name RVA	02A3 QueryPerformanceCounter
00000C18	000024C8	Hint/Name RVA	0239 IsDebuggerPresent
00000C1C	000024AA	Hint/Name RVA	034A SetUnhandledExceptionFilter
00000C20	0000248E	Hint/Name RVA	036E UnhandledExceptionFilter
00000C24	0000247A	Hint/Name RVA	0142 GetCurrentProcess
00000C28	00002466	Hint/Name RVA	035E TerminateProcess
00000C2C	00002448	Hint/Name RVA	0226 InterlockedCompareExchange
00000C30	00002440	Hint/Name RVA	0356 Sleep
00000C34	0000242A	Hint/Name RVA	0229 InterlockedExchange
00000C38	00002532	Hint/Name RVA	01CA GetSystemTimeAsFileTime
00000C3C	00000000	End of Imports	KERNEL32.dll
00000C40	0000235E	Hint/Name RVA	00D0 __p__fmode
00000C44	0000236C	Hint/Name RVA	016D _encode_pointer
00000C48	0000237E	Hint/Name RVA	00E6 __set_app_type
00000C4C	00002390	Hint/Name RVA	014E _crt_debugger_hook
00000C50	000023B2	Hint/Name RVA	03ED _unlock
00000C54	0000234E	Hint/Name RVA	00CC __p__commode
00000C58	000023CA	Hint/Name RVA	027C _lock
00000C5C	000023D2	Hint/Name RVA	0322 _onexit
00000C60	000023DC	Hint/Name RVA	0163 _decode_pointer
00000C64	000023EE	Hint/Name RVA	0176 _except_handler4_common
00000C68	00002408	Hint/Name RVA	0211 _invoke_watson

Viewing IMPORT Address Table

CFF Explorer

CFF Explorer VII - [fc_upx.exe]

File Settings ?

fc_upx.exe

- File: fc_upx.exe
  - Dos Header
  - Nt Headers
    - File Header
    - Optional Header
      - Data Directories [x]
  - Section Headers [x]
  - Import Directory
  - Resource Directory
  - **Address Converter**
  - **Dependency Walker**
  - **Hex Editor**
  - **Identifier**
  - **Import Adder**
  - **Quick Disassembler**
  - **Rebuilder**
  - **Resource Editor**
  - **UPX Utility**

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
UPX0	0000C000	00001000	00000000	00000400	00000000	00000000	0000	0000	E0000080
UPX1	00008000	0000D000	00007C00	00000400	00000000	00000000	0000	0000	E0000040
.rsrc	00001000	00015000	00000200	00008000	00000000	00000000	0000	0000	C0000040

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ▮.▮...▮...ÿÿ..

Basic Dynamic Analysis – concepts, tools and methods

- Sandbox
- SystemInternals
  - Process Monitor (ProcMon) → filesystem and registry Activity
  - Process Explorer (procexp.exe)
  - Handle.exe
  - Autoruns.exe

Dynamic Analysis – more tools

- Dependency Walker
- Regshot
- FakeNet / ApateDNS
- Wireshark (on Kali/Ubuntu)
- INetSim (on Kali/Ubuntu)
- rundll32.exe <DLL>,<Function>

Advanced Static Analys

We HAVE to know:

- C and Assembly
- Stack / Heap layout
- Windows Internals
- Function Calls
  - cdecl → caller cleans up the stack
  - stdcall → callee cleans up the stack
  - fastcall → two parameters passed in registers (EDX,ECX)
    remaining parameters passed through stack
    caller cleans up the stack

IDA Pro

IDA - Z:\Malware\Advanced\Malwares_Own\malware1.exe

File Edit Jump Search View Debugger Options Windows Help

No debugger

Functions window

Function name
- _main
- __security_check_cookie(x)
- ___tmainCRTStartup
- start
- ___report_gsfailure
- _amsg_exit
- __onexit
- _atexit
- sub_4014BB
- _XcptFilter
- __ValidateImageBase

Line 1 of 28

Graph overview

IDA View-A | Hex View-A | Structures | Enums | Imports | Exports

```
arg_4= dword ptr  8

cmp     [esp+arg_0], 4
jge     short loc_40100A

xor     eax, eax
retn

loc_40100A:
push    esi
mov     esi, [esp+4+arg_4]
mov     eax, [esi+4]
push    eax             ; lpLibFileName
call    ds:LoadLibraryA
cmp     eax, 0FFFFFFFFh
jz      short loc_40102E

push    offset ProcName ; "NRACSConfigure"
push    eax             ; hModule
call    ds:GetProcAddress
test    eax, eax
jnz     short loc_401033

loc_40102E:
or      eax, 0FFFFFFFFh

loc_401033:
mov     ecx, [esi+0Ch]
```

100.00% (-210,633) (193,17) 00000405 00401005: _main+5

Output window

```
Using FLIRT signature: Microsoft VisualC 2-10/net runtime
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
Hex-Rays Decompiler plugin has been loaded (v1.5.0.110408)
  License: 57-3B73-7AE4-E2 ESET spol. s r.o. (36 users)
  The hotkeys are F5: decompile, Ctrl-F5: decompile all.
  Please check the Edit/Plugins menu for more informaton.
---------------------------------------------------------------------------------------
Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)]
IDAPython v1.5.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
---------------------------------------------------------------------------------------
```

Python

AU: idle | Down | Disk: 7GB

IDA Pro

IDA - Z:\Malware\Advanced\Malwares_Own\malware1.exe

File Edit Jump Search View Debugger Options Windows Help

No debugger

Functions window | IDA View-A | Hex View-A | Structures | Enums | Imports | Exports

Function name
- _main
- __security_check_cookie(x)
- ___tmainCRTStartup
- start
- ___report_gsfailure
- _amsg_exit
- __onexit
- _atexit
- sub_4014BB
- _XcptFilter
- __ValidateImageBase
- __FindPESection
- __IsNonwritableInCurrentImage
- _initterm
- _initterm_e
- __SEH_prolog4
- __SEH_epilog4
- __except_handler4
- __setdefaultprecision
- sub_4016A1
- ___security_init_cookie
- _crt_debugger_hook
- _unlock
- __dllonexit
- _lock
- _except_handler4_common

Line 1 of 28

```
.text:00401043
.text:00401044 ; [0000000F BYTES: COLLAPSED FUNCTION __security_check_cookie(x). PRESS KEYPAD "+" TO EXPAND]
.text:00401053 ; ---------------------------------------------------------------------------
.text:00401053
.text:00401053 _pre_cpp_init:
.text:00401053                 push    offset loc_4014DF
.text:00401058                 call    _atexit
.text:0040105D                 mov     eax, dword_403360
.text:00401062                 mov     dword ptr [esp], offset dword_40302C
.text:00401069                 push    dword_40335C
.text:0040106F                 mov     dword_40302C, eax
.text:00401074                 push    offset envp
.text:00401079                 push    offset argv
.text:0040107E                 push    offset argc
.text:00401083                 call    ds:__getmainargs
.text:00401089                 add     esp, 14h
.text:0040108C                 test    eax, eax
.text:0040108E                 mov     dword_403028, eax
.text:00401093                 jge     short locret_40109D
.text:00401095                 push    8
.text:00401097                 call    _amsg_exit
.text:0040109C                 pop     ecx
.text:0040109D
.text:0040109D locret_40109D:                          ; CODE XREF: .text:00401093↑j
.text:0040109D                 retn
.text:0040109E ; [00000176 BYTES: COLLAPSED FUNCTION ___tmainCRTStartup. PRESS KEYPAD "+" TO EXPAND]
.text:00401214 ; ---------------------------------------------------------------------------
.text:00401214
.text:00401214 $LN31:                                  ; _pre_c_init
.text:00401214                 cmp     word ptr ds:400000h, 5A4Dh
.text:0040121D                 jz      short loc_401223
.text:0040121F
00000443 00401043: main+43
```

Output window

```
Using FLIRT signature: Microsoft VisualC 2-10/net runtime
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
Hex-Rays Decompiler plugin has been loaded (v1.5.0.110408)
  License: 57-3B73-7AE4-E2 ESET spol. s r.o. (36 users)
  The hotkeys are F5: decompile, Ctrl-F5: decompile all.
  Please check the Edit/Plugins menu for more informaton.
---------------------------------------------------------------------------------------
Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)]
IDAPython v1.5.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
---------------------------------------------------------------------------------------
```

Python

AU: idle | Down | Disk: 7GB

IDA Pro

```
.text:00401223 loc_401223:                         ; CODE XREF: .text:0040121Dj
.text:00401223          mov     eax, ds:40003Ch
.text:00401228          cmp     dword ptr [eax+400000h], 4550h
.text:00401232          jnz     short loc_40121F
.text:00401234          movzx   ecx, word ptr [eax+400018h]
.text:0040123B          cmp     ecx, 10Bh
.text:00401241          jz      short loc_40125E
.text:00401243          cmp     ecx, 20Bh
.text:00401249          jnz     short loc_40121F
.text:0040124B          cmp     dword ptr [eax+400084h], 0Eh
.text:00401252          jbe     short loc_40121F
.text:00401254          xor     ecx, ecx
.text:00401256          cmp     [eax+4000F8h], ecx
.text:0040125C          jmp     short loc_40126
```

mindthesec FORUM

Ida Pro

```
 sub_401000      proc near               ; CODE XREF: _main+4p
.text:00401000
.text:00401000 var_4           = dword ptr -4
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 push    ecx
.text:00401004                 push    0               ; dwReserved
.text:00401006                 push    0               ; lpdwFlags
.text:00401008                 call    ds:InternetGetConnectedState
.text:0040100E                 mov     [ebp+var_4], eax
.text:00401011                 cmp     [ebp+var_4], 0
.text:00401015                 jz      short loc_40102B
.text:00401017                 push    offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C                  call   sub_40105F
.text:00401021                 add     esp, 4
.text:00401024                 mov     eax, 1
.text:00401029                 jmp     short loc_40103A
```

Other details

- Handles (processes, modules, windows, file,etc)
- Filesystems
  - ReadFile/WriteFile
  - CreateFileMapping/MapViewOfFIle
- Registry
  - RegOpenKeyEx
  - RegSetValueEx
  - RegGetValue

Other details (cont)

- Mutexes
  - WaitForSingleObject
  - ReleaseMutex
- Services
  - OpenSCManager
  - CreateService
  - StartService
- COM Model
  - CoCreateInstance
  - IWebBrowser2

mindthesec FORUM

Advanced Dynamic Analysis

- Debuggers provide an information that is impossible to gather with static analysis
- Memory, registry, arguments, stack, and so on.
- Debugger programs:
  - Immunity
  - OllyDgb
  - WinDbg (user and kernel-mode code)

mindthesec FORUM

OllyDbg

OllyDbg - malware1.exe - [CPU - main thread, module malware1]

C File View Debug Plugins Options Window Help

Analysing malware1: 11 heuristical procedures, 17 calls to known functions

Paused

mindthesec FORUM

WinDbg


```
lkd> !heap -s
LFH Key                   : 0x000000d23b4b77ef
Termination on corruption : ENABLED
          Heap     Flags   Reserv  Commit  Virt   Free  List   UCR  Virt  Lock  Fast
                            (k)     (k)    (k)     (k) length      blocks cont. heap
-------------------------------------------------------------------------------------
Virtual block: 0000000003b10000 - 0000000003b10000 (size 0000000000000000)
Virtual block: 0000000004a10000 - 0000000004a10000 (size 0000000000000000)
0000000000090000 00000002    8192    6828   8192     36    18     4     2      0   LFH
0000000000010000 00008000      64       8     64      5     1     1     0      0
Virtual block: 0000000003890000 - 0000000003890000 (size 0000000000000000)
Virtual block: 0000000003dd0000 - 0000000003dd0000 (size 0000000000000000)
Virtual block: 0000000003700000 - 0000000003700000 (size 0000000000000000)
0000000000430000 00001002    7232    6940   7232     62    32     4     3      0   LFH
0000000001cd0000 00001002     512     104    512      1     5     1     0      0   LFH
0000000001df0000 00001002     512     112    512      8     8     1     0      0   LFH
00000000022c0000 00001002     512     180    512     30     3     1     0      0
-------------------------------------------------------------------------------------
lkd>
```

mindthesec FORUM

WinDbg – few useful commands

da → it reads from memory (ASCII)
du → it reads from memory (Unicode)
dd → it reads from memory (32-bits double words)
dx → display a C++ expression (for Windows 10)
bp → it sets a breakpoint
lm → it lists the modules loaded into process
u → disassemblies a function
x → it searches for funtions or symbols
ln → it lists the symbol for a given memory address
dt → it views a structure information
!drvobj → it views a drive object
!devhandles → it shows applications that have a handle to a device
! idt → it shows the Interrupt Descriptor Table (IDT) (stores the ISR information that is called each time that interruption code is produced)

Debuggers operations and breakpoints

- User-mode vs Kernel mode
- Single-Step / Step-Over / Step-Into
- Breakpoints → to stop the execution and examine the program state (registry, memory, flags, stack)

Breakpoints

- Software Breakpoints (F2 on Immunity/OllyDbg)
  - Alter the first byte of the instruction (0xCC)
  - Generate an exception
  - If the instruction's bytes change, the breakpoint will not ocurr
- Hardware Breakpoints
  - It don't matter which bytes are stored at the location.
  - Only fours hardware breakpoints
  - Easy to be modified by running program
- Conditional Breakpoints
  - It will break only if a certain condition is true
  - Implemented as software breakpoints

Anti-Forensic Methods

Multiple Techniques

- User-mode Rootkits
- Kernel-mode Rootkits
- DLL Load-Order Hijacking
- Asynchronous Procedure Call Injection (APC Injection)
  - Remember: APCs allow a program to run its code in the context of another thread

Multiple Techiques

- Process Injection
- DLL Injection
- Direct Injections
- Hook Injection
- Process Replacement

Multiple Techniques

- Anti-Debugging
- Anti-Dissassembly
- Anti-Virtual Machines
- Packers
- Crypto

GPU Malwares

GPU Malwares

- GPU (Graphics Process Unit)
- They are used to perform graphics rendering. Therefore, they offload the CPU during intensive operations.
- Do you remember about cracking passwords by using Oclhashcat tool (http://hashcat.net/oclhashcat/), which supports CUDA (Compute Unified Device Architecture) and OpenCL?

GPU Malwares

- CUDA is an extension of C language and a runtime library, which presents several hardware features that are not usually available through graphic APIs.
- CUDA was released by NVidia.

GPU Malwares

- Every application using CUDA serially runs a chunk of the program on CPU and another part name kernel (parallel) on GPUs.
- The kernel part must be called for a process running on CPU.

GPU Malwares

- DMA (Direct Memory Access) makes easier for CPU and GPU code to execute in a concurrent way.
- GPU Malware are composed by:
  - CPU Code
  - GPU Code (kernel)

GPU Malwares

- CUDA provides methods for exchanging data between CPU and GPU. For example, the CPU memory can be mapped into GPU address space (http://www.nvidia.com/object/cuda_home_new.html)
- As GPU malwares are linked to CUDA library, so the GPU code can be executed as a regular user (without administrative privileges).

GPU Malwares

- By using CUDA, GPU malwares are specific for NVidia processors. Nonetheless, as many vendor are using OpenCL (https://en.wikipedia.org/wiki/OpenCL) then most GPU malwares will execute on any video card.

GPU Malwares

- Packers (an anti-forensic technique), which insert the unpacker code inside the code, trend to become more complex (not only XOR obsfuscation) because GPU processors have more power than CPUs.
- Nowadays, specialized automatic extractors do not handle GPU packed malware.
- There is little information about dynamic and static analysis of malware running on GPUs.
- Virtual machines (VMware / VirtualBox) do not simulate GPU video boards, so it is more difficult to make tests.

GPU Malwares

- The GPU malware process is described by the following steps:

0. The executable contains the malware and the decryptor.
1. The initial code runs on CPU, loads the kernel on GPU and allocates mapping memory for the packet malware
2. This mapped memory allows that GPU code can be also accessed by the memory area on CPU. Thus, CPU and GPU share the same data.

GPU Malwares

3. The flow control is transfered to GPU, which runs the unpacker/decryptor (transfered by the initial code).
4. The decryptor routine decrypts the binary (that is in the mapped memory) and the control is tranfered to CPU for running this unpacked code.
5. This process leaves little footprint in the CPU environment to be analyzed.

GPU Malwares

- Is it possible to analyze the malware in the memory after the decryption process ?
- Yes, we could acquire the memory right after the decryptor finishing its job.
- However, things are not easy. The decrytor only decrypt some instructions, run and encrypt them again.
- It could be not feasible with CPUs, but GPUs encrypt and decrypt anything very fast.

GPU Malwares

- Each instruction can be encrypted by a different key.
- However, each key is stored in a private memory on video board that is not available to CPU.
- GPUs can use other anti-forensic techiques such as code checksumming that prevents any code modification by analysts. The checksum is calculated by GPUs and many these operations can be done with different parts of code in parallel (similar to Skype code)

mindthesec FORUM

GPU Malware and Tools: Are they already exist?

Examples and Tools

- Win_Jelly (a RAT for demonstration purposes)
- Demon (a keylogger for Linux)
- JellyFish (a GPU malware for Windows)
- Tools: There is neither any anti-virus nor tool for analysis. Everything must be manually done.

Conclusion

Conclusion

- Nowadays, there is no solution.
- But there are researchers working on GPU Malwares, so there is hope...

mindthesec FORUM

Thank you!

Alexandre Borges

e-mail: alexandreborges@alexandreborges.org

LinkedIn: www.linkedin.com/in/aleborges

Twitter: @ale_sp_brazil

Blog: http://alexandreborges.org

http://alexandreborges.org

Next year, I will be releasing courses about:

- Malware Analysis (basic and advanced)
- Memory Forensic Analysis (1 and 2)
- WinDbg
- Hacking (1, 2 and 3)
- Forensic Analysis