

Binary Analysis Class 101

N KOREAN
HACKERS
AT WORK

Disclaimer

- Artifacts are weaponized
 - Pwd 'infected'
- Handle with care
- Some artifacts may contain offensive language!

Learning Outcomes

- Binary analysis fundamentals
- Static analysis
 - Tools
 - Hands-on binary analysis
- Analysis Results
 - IoCs
 - VT reports
- PE Structure

Not Covered

- Disassembly – IDA Pro
- Assembly – Instructions, registers etc
- Dynamic analysis and debugging
- Programming constructs

Prerequisites

- Windows >7 x64 (Ideally in VM)
- Min 4GB RAM/50GB HDD
- NIC disabled / No Internet
- Analysis apps provided on USB

Topics

- TLP/TI/OPSEC
- Safe handling
- What is Binary Analysis
- artifacts – A Primer
- Analysis Methodology
- Tools
- Challenges
 - Demo walkthrough
 - Practical 0 - Starting Point
 - Practical 1 – Pwned for Eternity
 - Practical 2 – Go Large
 - Practical 3 – Blizzard
 - Practical 4 – Scream
 - Practical 5 – Scream Forever
 - Practical 6 - Sunshop
 - Practical 7 – Smoke ‘em out
- Walkthrough – Advanced - APT29

Terms

- **Artifact** = binary, sample etc
- **RE** = malware/binary analysis etc
- **Adversary** = threat actor, attacker, developer etc

Binary Analysis

- What is it?
 - Static – inspection, no runtime analysis
 - Dynamic – runtime analysis, behaviors
 - Hybrid analysis – debugging code / follow code path
- Why do it? Gather information
 - Incident Response
 - Impact to the system
 - Last resort – this is a reactive scenario i.e. containment and recovery
- Resource intensive
 - AV and TI can save effort
 - Manual malware analysis is a last resort – automation helps

Challenge

- Artifacts designed to achieve an objective as defined by the adversary
 - Financial – ransomware / info stealer / mining
 - Destructive – virus / wiper
 - Intelligence – APT
 - Other - proxy
- artifacts need to execute instructions to achieve the objective
 - Typically in binary or script form
- artifacts can take many forms – to defeat PSPs / Checkers
 - Containers - ISO
 - Packed – compressed / self extracting

Artifacts – Many Forms

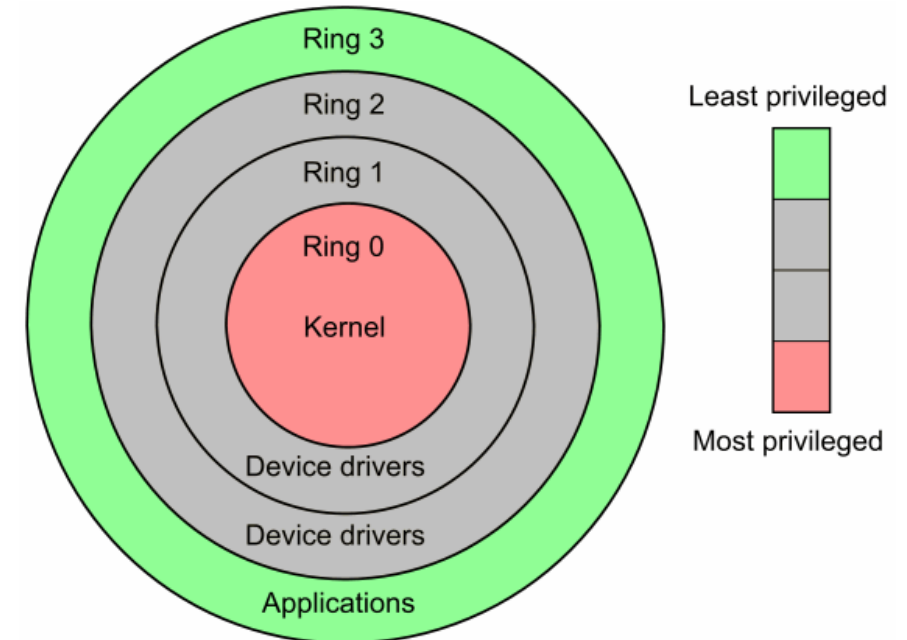
- Binary = exe, com, dll, scr, drv, sys, ocx, jar, class
- Scripts = ps, bat, js, vba
- Application = doc, rtf, docx, pptx, pdf etc
- Compilers = .Net, C++, AutoIT etc
- Installers = msi, etc
- Exploits = Ink, etc
- Others = web code, email, browser objects etc
- Operate at user or kernel mode (rootkit)

Malware

- Different forms:
 - Worms
 - Virus
 - Remote Access Trojan
- Don't have to be malicious – despite name
 - Could just simply harvest system information (recon), act as backdoor etc
- Could be multiple stages
 - Smaller files – evade monitoring/detection
 - Different technology i.e. doc -> exe
 - 1st stage (delivery), 2nd stage (dropper) etc

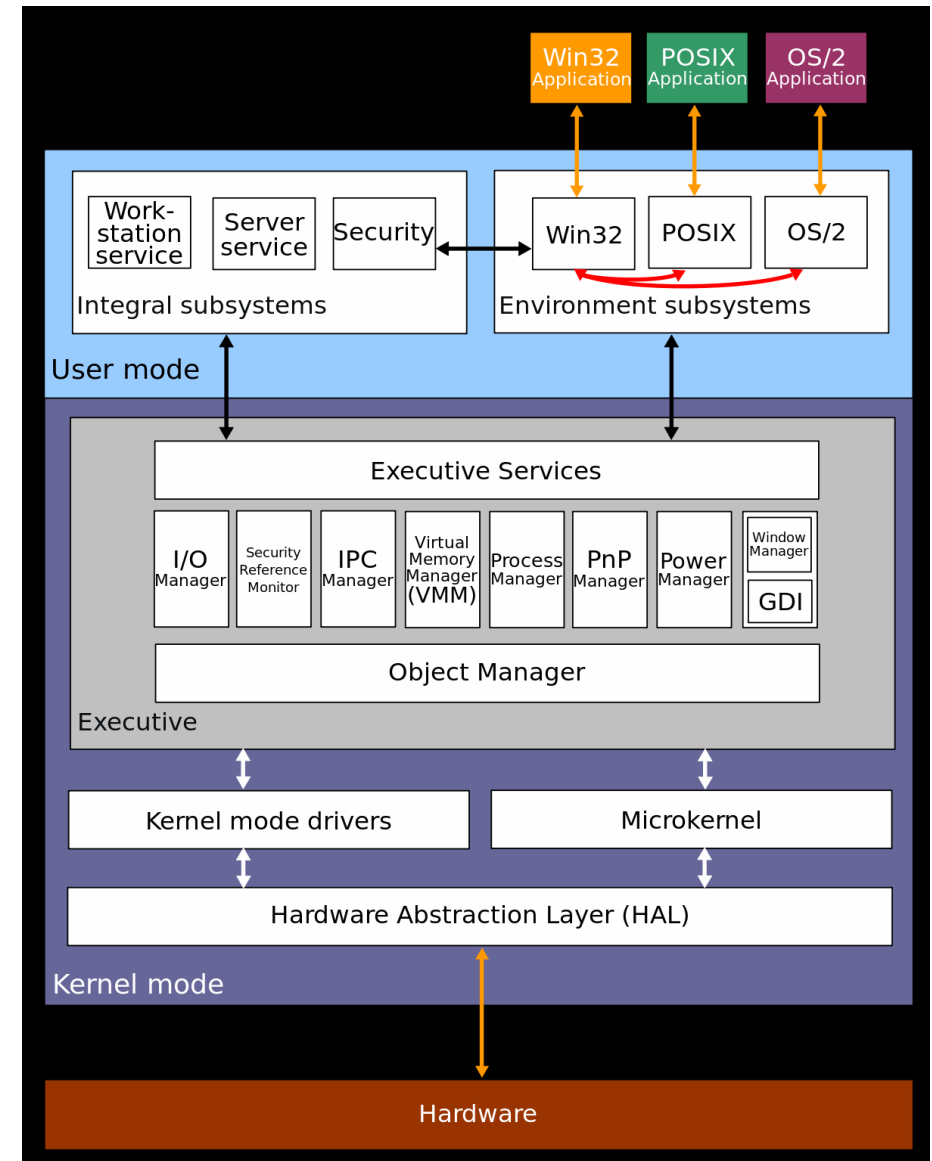
Malware Execution (Two Modes)

- Kernel
 - the executing code has complete and unrestricted access to the underlying hardware
 - System level access – can hide, access priv resources
- User
 - the executing code has no ability to directly access hardware or reference memory.
 - Code running in user mode must delegate to system APIs to access hardware or memory
 - Most apps run at this level



Modes Pt 2

- There is a -1 ring layer – if referring to the hyper-visor layer
 - Not an official number
- Getting to ring layer 0 enables software to hide from AV
 - Rootkits
 - Hook into system calls – data manipulation/process hiding
 - Risk of causing BSOD/kernel panic
- Getting harder to achieve
 - OS protections – kernel patch protection, signed drivers etc
 - Still can be exploited – APT28 exploited a VirtualBox NIC driver...
 - <https://github.com/hfiref0x/TDL> [Turla Driver Loader]



Focus

- Binary = Windows PE
- User mode
- Static analysis

Tools

- Inspection
 - TrID – identify file types, CLI
 - PE Studio – structure analysis, string extraction, header information
 - CFF Explorer – Like PE Studio, provides binary and address references
 - Resource Hacker – Embedded objects, dialogs, windows etc
 - Strings – ASCII + Unicode chars
 - EXEInfo – Packer identification
- Unpackers
 - 7 zip – handles many containers/compressed files
 - UPX – Pack and unpack binaries
- Disassembly (not covered)
 - IDA Pro – low level disassembly, and analysis/debugger. Many CPU architectures supported
 - Ghidra – NSA version of IDA Pro
- De-compilers
 - ILSpy - .Net de-compiler
 - DotNotPeek – Another .Net de-compiler
- Debuggers
 - OllyDbg – No longer supported
 - X64dbg – Supports 32 bit and 64 bit

PE File Primer

- PE file could be EXE, DLL, SCR, DRV etc.
- Headers (x3) followed by tables (x2)
- Key Components:
 - DOS Header
 - Common Object File Format (COFF)
 - Portable Executable (PE) Header (Optional Header)
 - Data Directory Table
 - Section Table

DOS Header

- Contains the magic number (MZ), the DOS stub, and some very low-level information about the executable.

Common Object File Format (COFF) Header

- Contains machine type (e.g. i386 or Intel64) and some basic flags about the type of executable that this file is.

Portable Executable (PE) Header

- Contains information about the runtime of the process, e.g. entry point address, code and data sizes, base addresses, the Windows subsystem to run in (e.g. GUI, console, driver, EFI, etc.), the characteristics of the module (e.g. is it NX / ASLR compatible?), the initial stack and heap sizes, etc.

Data Directory Table

- A set of pointers (RVAs) and sizes for sixteen "special" segments of the file data, e.g. the import / export directories, the debug directory, .NET metadata, security directory, relocation table, TLS directories, etc


Section Table

- A table that represents various sections within the executable. Sections contain the data that constitutes the actual program, e.g. the instructions.

How Does It Work?

Member	Offset	Size	Value	Meaning
Magic	00000098	Word	010B	PE32
MajorLinkerVersion	0000009A	Byte	08	
MinorLinkerVersion	0000009B	Byte	00	
SizeOfCode	0000009C	Dword	0000DA00	
SizeOfInitializedData	000000A0	Dword	00000800	
SizeOfUninitializedData	000000A4	Dword	00000000	
AddressOfEntryPoint	000000A8	Dword	0000F95E	.text
BaseOfCode	000000AC	Dword	00002000	
BaseOfData	000000B0	Dword	00000000	
ImageBase	000000B4	Dword	00400000	
SectionAlignment	000000B8	Dword	00002000	
FileAlignment	000000BC	Dword	00000200	
MajorOperatingSystemVers...	000000C0	Word	0004	
MinorOperatingSystemVers...	000000C2	Word	0000	
MajorImageVersion	000000C4	Word	0000	
MinorImageVersion	000000C6	Word	0000	
MajorSubsystemVersion	000000C8	Word	0004	
MinorSubsystemVersion	000000CA	Word	0000	

PE Format

08/26/2019 • 126 minutes to read •  +2

This specification describes the structure of executable (image) files and object files under the Windows family of operating systems. These files are referred to as Portable Executable (PE) and Common Object File Format (COFF) files, respectively.

16	4	AddressOfEntryPoint	The address of the entry point relative to the image base when the executable file is loaded into memory. For program images, this is the starting address. For device drivers, this is the address of the initialization function. An entry point is optional for DLLs. When no entry point is present, this field must be zero.
20	4	BaseOfCode	The address that is relative to the image base of the beginning-of-code section when it is loaded into memory.

PE Files

c:\projects\binary analysis 101		
indicators (5/13)		
virustotal (47/57)		
dos-header (64 bytes)		
dos-stub (!This program c		
file-header (Jul.2015)		
optional-header (GUI)		
directories (5)		
sections (99.12%)		
libraries (mscoree)		
imports (_CorExeMain)		
exports (0)		
tls-callbacks (n/a)		
resources (2)		
strings (32/716)		
debug (n/a)		
manifest (invoker)		
version (Setting.exe)		
certificate (n/a)		
overlay (n/a)		
property	value	
name	.text	
md5	F73AF5CD30F6ED2308A1A3...	
file-ratio (99.12 %)	95.61 %	
file-cave (656 bytes)	156 bytes	
entropy	5.578	
raw-address	0x00000200	
raw-size (57856 bytes)	0x0000DA00 (55808 bytes)	
virtual-address	0x00402000	
virtual-size (57200 bytes)	0x0000D964 (55652 bytes)	
entry-point (0x0000F95E)	x	
writable	-	
executable	x	
shareable	-	
discardable	-	
initialized-data	-	
uninitialized-data	-	
readable	x	
self-modifying	-	
blacklisted	-	

```
TrID/32 - File Identifier v2.24 - (C) 2003-16 By M.Pontello
Definitions found: 11814
Analyzing...
```

```
Collecting data from file: sample(1).bin
55.8% (.EXE) Generic CIL Executable (.NET, Mono, etc.) (73294/58/13)
21.0% (.EXE) Win64 Executable (generic) (27624/17/4)
9.9% (.SCR) Windows screen saver (13101/52/3)
5.0% (.DLL) Win32 Dynamic Link Library (generic) (6578/25/2)
3.4% (.EXE) Win32 Executable (generic) (4508/7/1)
```

Finding IoCs

!This program cannot be run in DOS mode.

.rsrc

Setting.exe

avicap32.dll

Control

System.Net

Copy

Start

Kill

Delete

Shell

Write

Replace

8.0.0.0

My.Application

4.0.0.0

10.0.0.0

adobeupdate.sytes.net

adobeupdate.exe

Software\Microsoft\Windows\CurrentVersion\Run

Start

.exe

WScript.Shell

.Ink

Software\

Replace

Write

file-type	executable
date	empty
language	neutral
code-page	Unicode UTF-16, little endian
CompanyName	Microsoft
FileDescription	Setting
FileVersion	1.0.0.0
InternalName	Setting.exe
LegalCopyright	Copyright © Microsoft 2015
OriginalFilename	Setting.exe
ProductName	Setting
ProductVersion	1.0.0.0
Assembly Version	1.0.0.0

C:\ProgramData

software\microsoft\windows\currentversion\run

taskmgr

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer

shutdown -r -t 00

Connect

.tmp

netsh firewall delete allowedprogram "

cmd.exe /k ping 0 & del "

netsh firewall add allowedprogram "

melt.txt

CapsLock

[ENTER]

/c start

Software\Classes\

Setting.exe

Setting.exe

GetVolumeInformation

GetVolumeInformationA

GetKeyboardState

MapVirtualKey

GetKeyboardLayout

GetAsyncKeyState

FindWindow

FindWindowA

GetForegroundWindow

GetWindowText

De-compile

```
[DllImport("user32", CharSet = CharSet.Ansi, EntryPoint = "FindWindowA", ExactSpelling = true, SetLastError = true)]
private static extern long FindWindow([MarshalAs(UnmanagedType.VBByRefStr)] ref string lpClassName, [MarshalAs(UnmanagedType.VBByRefStr)] ref string lpWindowName);

private bool IsVmWare()
{
    string lpClassName = "VMDragDetectWndClass";
    string lpWindowName = null;
    long num = FindWindow(ref lpClassName, ref lpWindowName);
    if (num == 0)
    {
        return true;
    }
    return false;
}
```



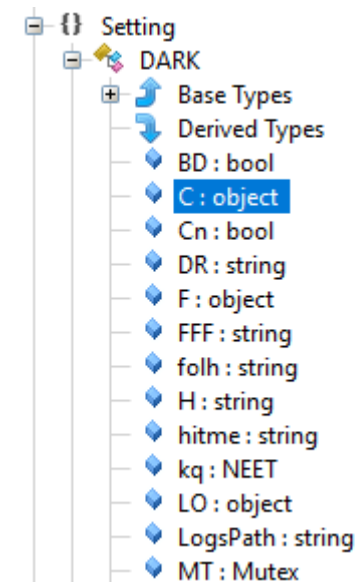
Dark .NET

موقوف لمخالفة الشروط



Visual Basic كود

```
1 Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As St
2 Private Function IsVmWare() As Boolean
3     Dim Hnd1 As Long
4     Hnd1 = FindWindow("VMDragDetectWndClass", vbNullString)
5     If Hnd1 = 0 Then
6         IsVmWare = True
7     Else
8         IsVmWare = False
9     End If
10 End Function
```



Programming

- Language = C, C++, C#, VB.Net, Delphi, etc
 - Each language has own syntax, libraries etc. Fairly human readable
 - Fundamentally, all use the same underlying OS API to access functions i.e. input, display, GUI, file I/O etc.
- Compile into machine code (object code)
- Linker
 - Turns object code into executable
- Note:
 - Sometimes there are 'traces' in the exe that indicate the language used to program
 - Peid and other binary browsers can identify the programming language

Encoding

- Bypass content checkers
- Makes RE more difficult
- Known as packing
- Either encrypted or compressed sections, which raises the entropy for the specific section.

```
38.2% (.EXE) UPX compressed Win32 Executable (27066/9/6)
37.5% (.EXE) Win32 EXE Yoda's Crypter (26569/9/4)
 9.2% (.DLL) Win32 Dynamic Link Library (generic) (6578/25/2)
 6.3% (.EXE) Win32 Executable (generic) (4508/7/1)
 2.8% (.EXE) OS/2 Executable (generic) (2029/13)
```

projects\binary analysis 101	property	value	va
indicators (15/28)	name	UPX0	UI
virustotal (52/68)	md5	n/a	FA
dos-header (64 bytes)	file-ratio (99.67 %)	0.00 %	99
dos-stub (!This program c	file-cave (0 bytes)	0 bytes	0
file-header (Jun.2012)	entropy	n/a	7.1
optional-header (GUI)	raw-address	0x00000400	0x
directories (4)	raw-size (311296 bytes)	0x00000000 (0 bytes)	0x
sections (entry-point)	virtual-address	0x00401000	0x
libraries (2)	virtual-size (2596864 bytes)	0x0022D000 (2281472 bytes)	0x
imports (1/7)	entry-point (0x002797A0)	-	x
exports (0)	writable	x	x
tls-callbacks (n/a)	executable	x	x
resources (unknown)	shareable	-	-
strings (1/4722)	discardable	-	-
debug (n/a)	initialized-data	-	x
manifest (invoker)	uninitialized-data	x	-
version (n/a)	readable	x	x
certificate (n/a)	self-modifying	x	x
overlay (n/a)	blacklisted	x	x

Embedded

- Icon
- Certificate
- DLLs

Finding artifacts

- Online repositories
 - Any.run
 - VT
 - Hybrid Analysis
 - GitHub TheZoo
- Hunt
 - PasteBin
 - Torrent
 - Usegroups
 - Dark Web
- Build your own

Safe Handling

- artifacts should be zipped and password protected
 - Common password is 'infected'
- Defang when communicating IP, URL, etc
- Application whitelisting
- Don't leave artifacts lying around – secure
- Create an analysis VM

Methodology

- Identify – Static analysis
 - Characterise - detect
 - IoCs – strings, size
 - Functions and capabilities
- Code Inspection – Static and Dynamic
 - Disassembly
 - Decompile
- Dependencies – Static and Dynamic
 - C2
 - Networking

Analysis Environment

- Windows VM
 - Snapshot
 - AV disabled
 - Telemetry
 - Fingerprints
 - FW
- VPN or TOR

Tooling - Pitfalls

- Understand tool capabilities – Read the manual
- Watch out for ‘execution’ features
 - PDF extract tool has an inspect – but actually executes extracted shellcode!
- Internet capabilities
 - PEid and others, interrogate online resources i.e. hash look ups etc.

Challenges

Practical – Demo

Demo

- artifact: 6EDB529B0DD6BF8BF0BEE10BE208D6584FA338C6AA6434EC565D574BB43D440C
- Compiler Type?
- Linker version?
- Internal name?
- Company name?
- Product name referenced?
- Method of persistence?
- Settings changed?
- Is it safe?

Practical – Starting Point

Starting Point

- artifact: 4249278855C637212BA55288480802812F5E194CAAD0EF4443965F42DB24CC4F
- Compiler Type?
- Internal name?
- Function?
- What Russian web browser is targeted?

Practical – Pwned for Eternity

Pwned for Eternity

- artifact: 85B936960FBE5100C170B777E1647CE9F0F01E3AB9742DFC23F37CB0825B30B5
- Name of the malware?
- What is the signature / compiler used?
- Capabilities?
- What is the result if backdoor installed?
- Does this have a GUI?
- Would it run? Why?

Practical – Go Large

Go Large

- artifact: d233335ee3810e1df0bcc768c283a122b2fbf7c322205098cccf1627be9b4e5d.bin
- Name of the malware?
- What is the signature / compiler used?
- Capabilities?
- What is the user agent string used?
- What is the second stage filename called?
- Is this malicious?

Practical – Blizzard

Blizzard

- artifact:
DC5E401C53B6CA8A92D72E6FBECEA52F6EF9D45C1E8D53902A6F674EA7F9EE16.bin
- Compile date?
- How many kernel.dll functions are imported?
- What entity was the target?
- What IoC could be used to check if present on host?
- Anything odd?
- Is this malicious?
- Threat actor?

Practical – Scream

Scream

- artifact:
9C3E13E93F68970F2844FB8F1F87506F4AA6E87918449E75A63C1126A240C70E.bin
- Signature?
- Beacon Ips?
- Anything odd?
- Is this malicious?
- Type or malware?
- Threat actor & target?

Practical – Scream Forever

Scream Forever

- artifact: *4D4B17DDBC4CE397F76CF0A2E230C9D513B23065F746A5EE2DE74F447BE39B9.bin*
- Beacon Ips?
- Capabilities?
- Resource language?
- Dialog version?
- Is this malicious?
- Type or malware?
- Threat actor & target?

Practical – Sunshop

Sunshop

- artifact: *9C3E13E93F68970F2844FB8F1F87506F4AA6E87918449E75A63C1126A240C70E.bin*
- File type?
- Name one of the program databases?
- Web IOCs?
- How many embedded PE files?
- Resource language?
- Capabilities?
- Is this malicious?
- Threat actor?

Practical – Smoke ‘em Out

Smoke 'em out

- artifact: *d2ee07bf04947cac64cc372123174900725525c20211e221110b9f91b7806332.bin*
- What is the file description?
- What is the internal/original name?
- What is the file type?
- Capabilities?
- Call-back domain?
- Token used for RC4?
- Is this malicious?

Wrap Up

- Would you hack back?
- Good resource:
- <https://docs.microsoft.com/en-us/windows/win32/debug/pe-format>

What Next

- Disassembly using IDA Pro / Ghidra
- Dynamic analysis
- Memory forensics