Reverse engineering malware analysis

Agenda

But why?

Lab Creation

The Malware Analysis Methodology

Code Reversing Framework

Reporting

Windows Architecture Primer / Structure***

Assembly Language Primer*** (video)

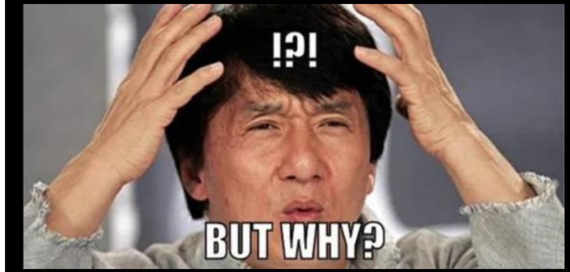
Signatures (AV, Packers, YARA)

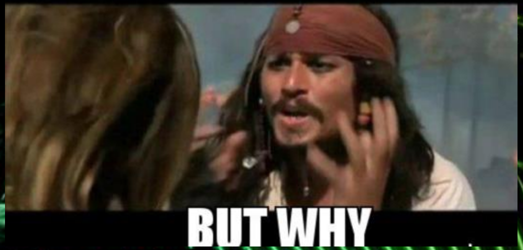
Radare

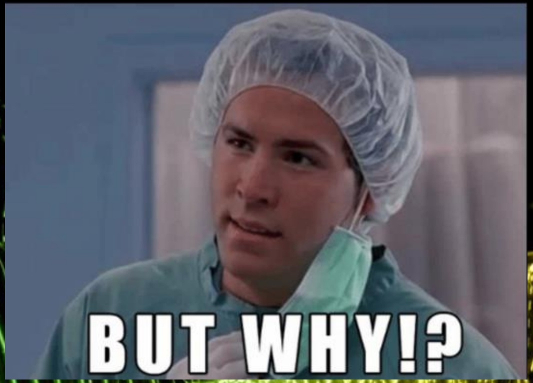
Examples

First things first: But why analyze malware?

- Analyze malware so you can:
  - Determine the nature of malware threats and asses the damage
  - Identify the scope of the incident
  - Determine the sophistication level of an intruder
  - Identify a vulnerability
  - Answer questions…
- Technical Questions:
  - What are the malware characteristics?
  - What are the Network and Host-based Indicators?
  - What is the Persistence Mechanism?
  - What is the Date of Compilation?
  - Is it packed?
  - When was it installed?
  - Does it have any rootkit functionality?
  - Was it designed to evade detection and thwart analysis?



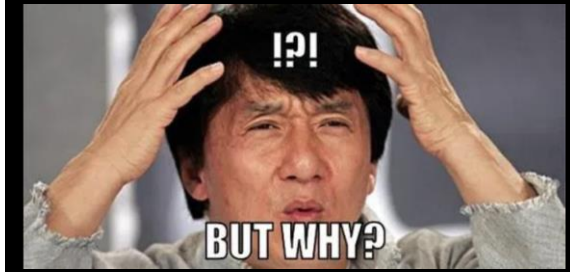


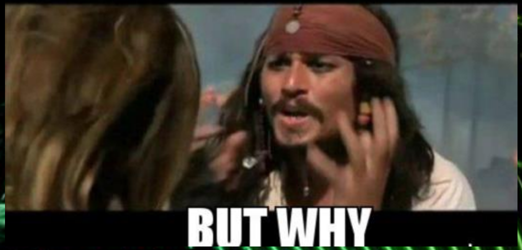


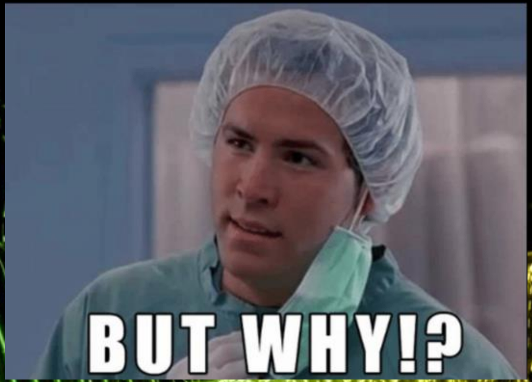
You can never have enough "But why's"!!!

First things first: But why?

- Business Questions:
  - What is the purpose of the malicious code?
  - How did it get into the environment?
  - How can you eradicate it?
  - Who is targeting the company and their level of sophistication and efficiency?
  - What was stolen?
  - How long have the intruders been in the network?
  - How can it be found on other machines?
  - Does it spread on its own?
  - What can be done in order to prevent this from happening in the future?







You can never have enough “But why’s”!!!

Lab Creation

- To analyze malware safely and effectively we need a properly configured lab
- In order to create a lab we need to understand the defense mechanism employed by adversaries in order to avoid their code to be detected and analyzed.
- In order to protect itself, the malicious code will try and detect and bypass the presence of:
    - a virtualize environment
    - debugging/disassembling software
    - anomalies in the system
    - Other
- Another model used for identifying and classifying malware according to how difficult is to detect is called **"Stealth Malware Taxonomy"**
- The design of a lab can be influenced by the **goals** and **specialization** of malware

Lab Creation

- Physical vs Virtualized environment
  - A physical environment simulates better but is costly ($ & time) to implement and maintain
  - A virtual environment is easier to maintain, but needs to be hardened and properly configured for the analysis to be efficient
- Virtualized environment is the preferred method
- Lab Components:
  - VM Host and/or Virtualization Server
  - “The Victim“
  - Lab Services
  - Network Hub
  - Honeypot
- Need to understand the methodology’s “drawbacks” – **Detection**, **Cost**, **Flexibility** and **Network Isolation**

Virtual Machines Environments

01 Native Virtualization - VMWare Workstation, Virtual PC

02 Paravirtualization - Parallels Workstation

03 Emulation -QEMU, Bochs

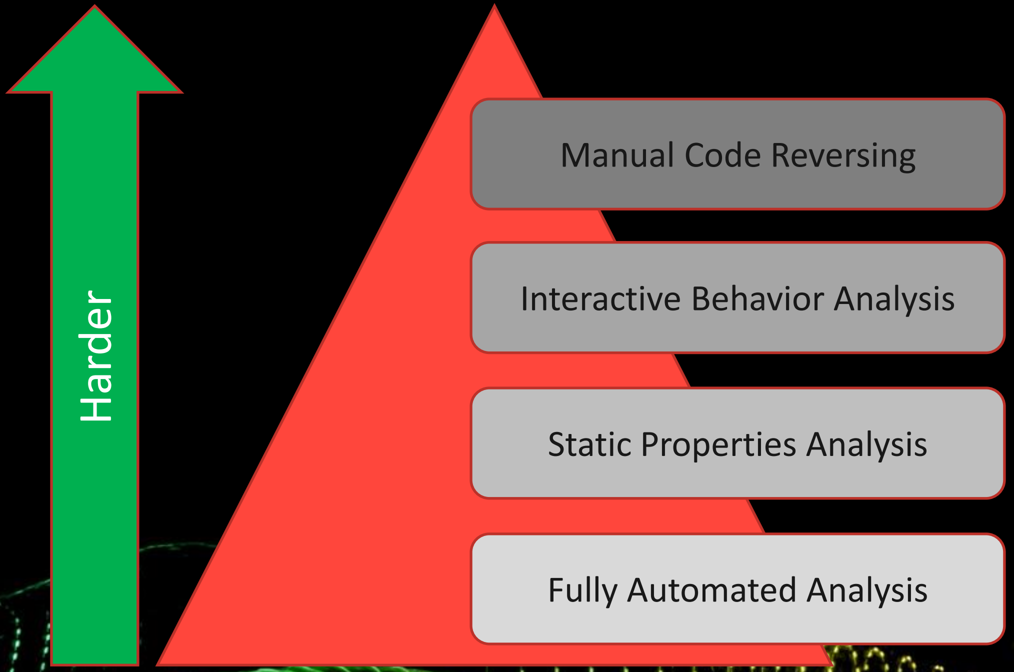
04 Dynamic Recompilation

05 Dedicated Virtualization Server - VMWare ESX

The Malware Analysis Framework

- The process of analyzing malicious software involves several stages, which can be listed in order of increasing complexity

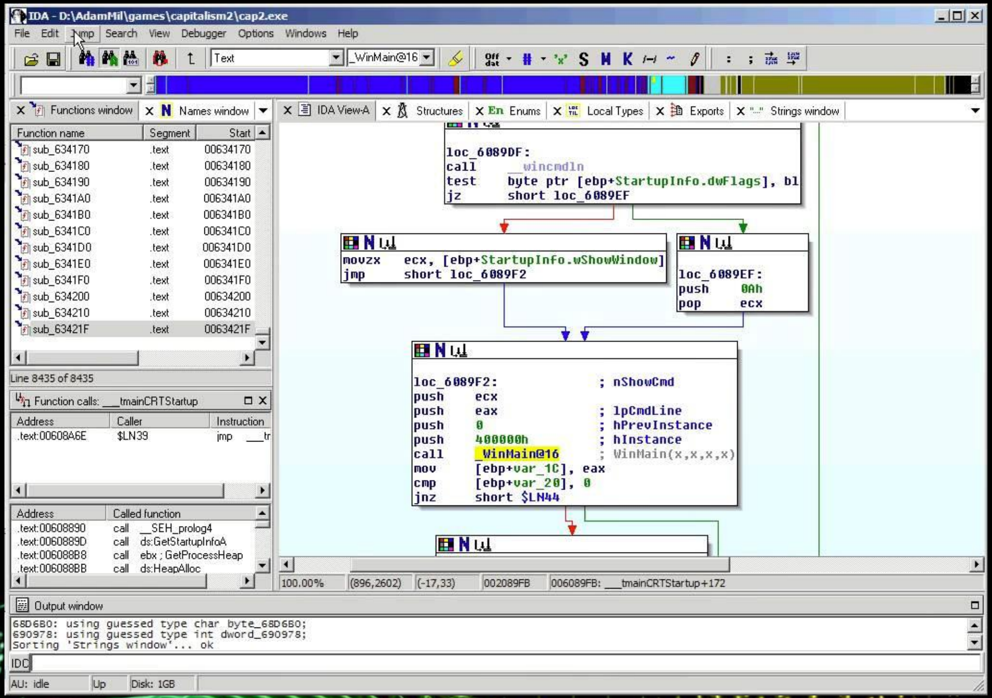
1. **Automatic Analysis**
   - The easiest way to begin investigating a specimen is to examine it using fully automated tools. These usually do not provide the same insight as a human analysis would, but contribute to the IR process by rapidly handling vast amounts of specimens
2. **Static Properties Analysis**
   - The next step would be to look at the static properties, also called metadata. This process entails examining the embedded strings, the overall structure and header data of the file, without running the program.



The Malware Analysis Framework

3. **Interactive Behavioral Analysis (Dynamic Analysis)**
    - The next step is running the specimen inside an isolated environment in order to observe its behavior. With the help of various monitoring tools (network, file, registry, processes, etc.), the analyst focuses on the capabilities and tactics employed by the program.
4. **Code Analysis (Disassemblers & Debuggers)**
    - When there are no more activities detected during the behavioral analysis, the next step is to start the code analysis phase. Code analysis enables the analyst to determine what are the specimen's capabilities by focusing on the assembly instructions.

    - **Static code-level analysis**
    - **Dynamic code-level analysis**



Code Reversing

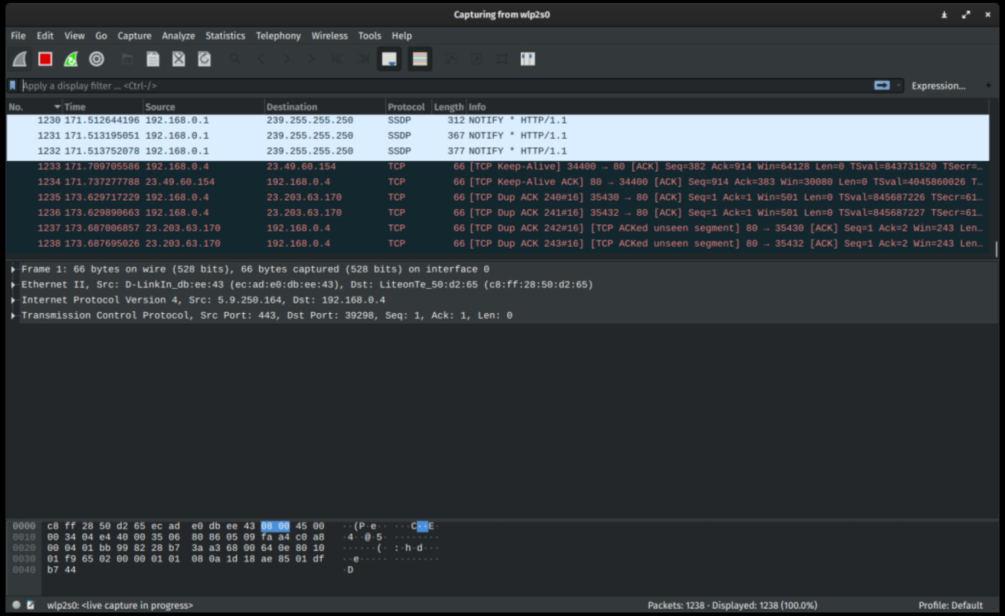
- Definition: Is the process of taking a captured executable (a stand-alone executable or a library file, such as a DLL) and deconstructing it in order to reveal:
  - its designs,
  - architecture
  - to extract knowledge from the object
- <u>**Benefits**</u>:
  - **Gain a deeper and more thorough understanding of Applications and Operating Systems**
  - **Develop, hone and improve Forensic Malware Analysis skills**
  - **Better prepared as a Forensic Analyst and Incident Response Handler**
  - **Other**

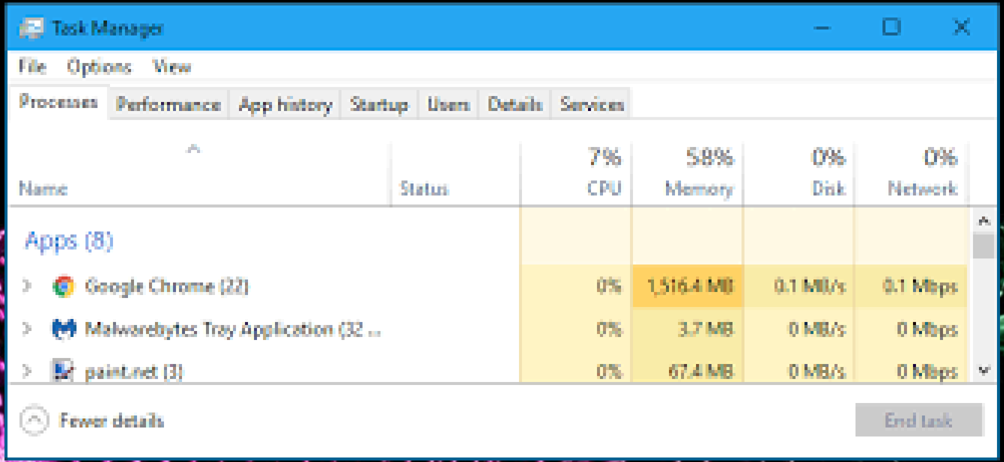
Framework

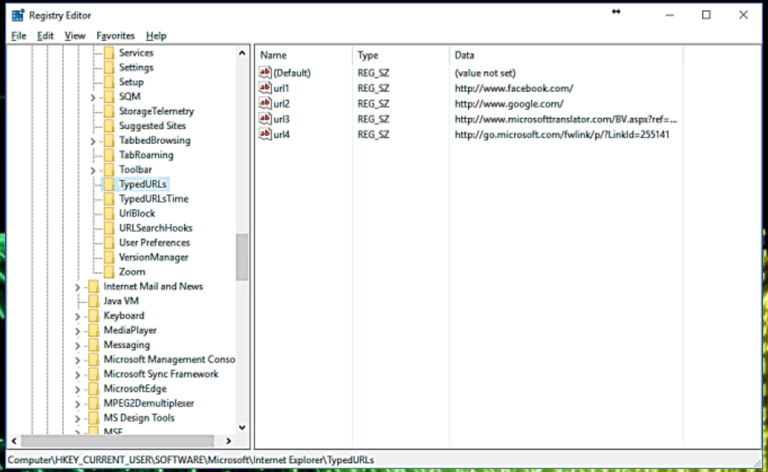
- Reverse engineering is used in both:
  - **Malware Development**
  - **Malware Delivery**
- Low Level Software:
  - Even if the source code is not available, the **low-level code** always is!
  - Assembly Code vs Machine Code
- **Framework** – the reverse process can be broken down into two steps**:**
  - **Code level** – extract the software's code concepts and algorithms from the machine code. (**IDA Pro, OllyDbg**, etc.)
  - **System level** – running tools to obtain information about the software, inspect the program and it's executable and track the program's input and output. (**SysInternals Suite, lsof, Wireshark**, etc.)
  - **Above steps are independent form one another, but, depending on the situation, it's recommended to view them as complementary.**

What to look for

- Important activities to follow and be aware of:
  - Registry Activity
  - File Activity
  - Process Activity
  - Network traffic







Reporting

- An important part of the analysis process is the “**reporting**” part
- Is essential to correctly and efficiently report your findings and results in order to better interact with other security professionals

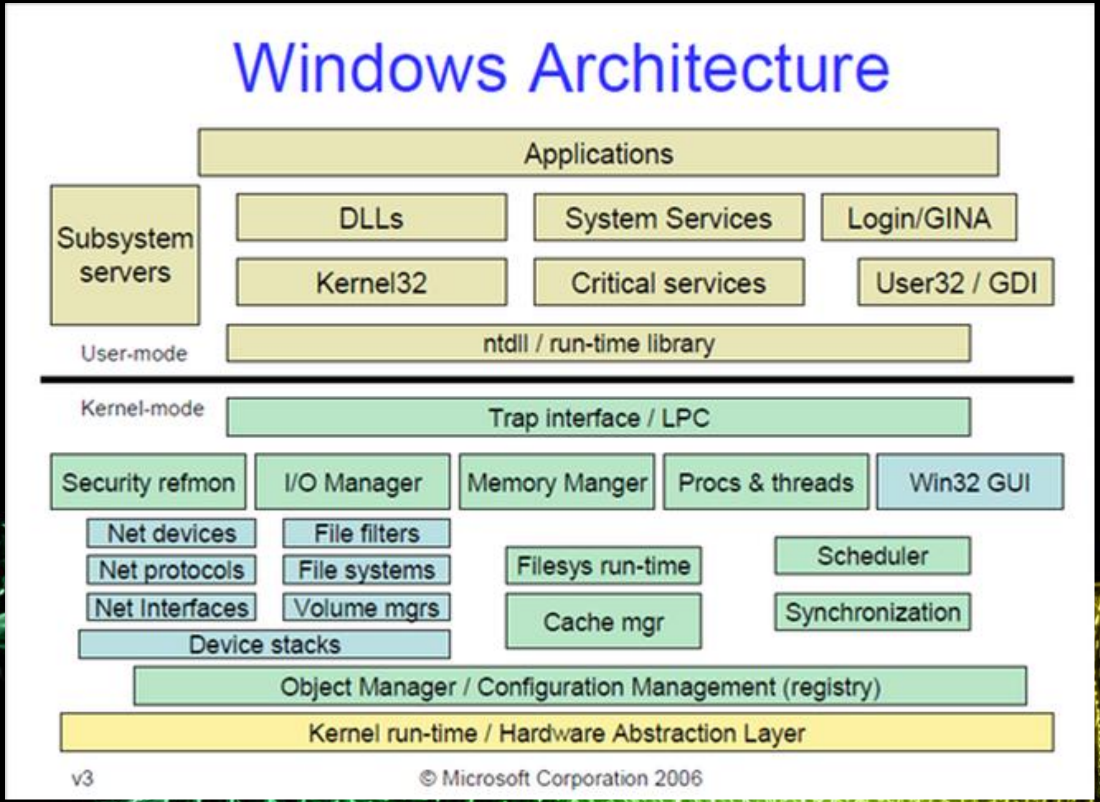
- **Intake**
  - Verbal reports
  - Suspicious samples
  - File system image
  - RAM image
  - Network logs
  - Anomaly observations
- **Product**
  - What malware does
  - How to identify it
  - The profile of the adversary
  - Reports and IOCs
  - Incident Response recommendations
  - Malware trends

Reporting

- The structure of a formal report should contain the following:
  - Summary of the analysis – key takeaway regarding the specimen's nature, origin and capabilities
  - Indicators of Compromise (IOCs) – type of file, name, size, hash, malware names (if known), and AV detection capabilities
  - Characteristics - the specimen capability to:
    - Infect
    - Lateral movement
    - Exfiltrate data
    - Create persistence
    - Interact with the adversary
  - Dependencies – what are the conditions that need to be met for the specimen to run
  - Behavioral and code analysis results
  - Incident Response Recommendations

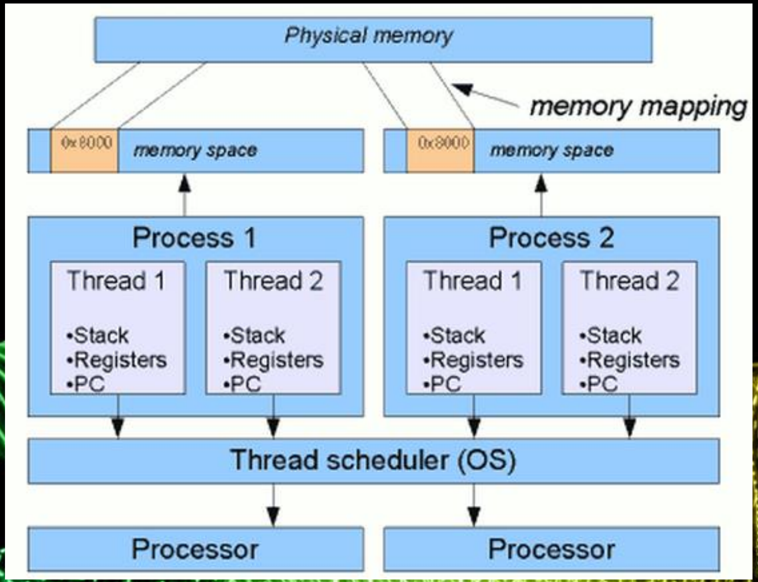
Windows Architecture Primer

- Why? – the need to better understand the inner workings of the Windows operating system in order to understand how malware can use and manipulate it.
- Topics:
  - **Kernel v User Mode -** enables the processor to enforce rules on how memory will be accessed.
  - **Paging -** physical memory is faster and more expensive than space on the hard drive.
  - **Objects –** The Windows OS manages objects (sections, files, and device objects, synchronization objects, processes and threads) using a centralized object manager component.
  - **Question: What is the difference in how objects are accessed between the Kernel and any Applications (at User-Mode level)?**



Windows Architecture Primer

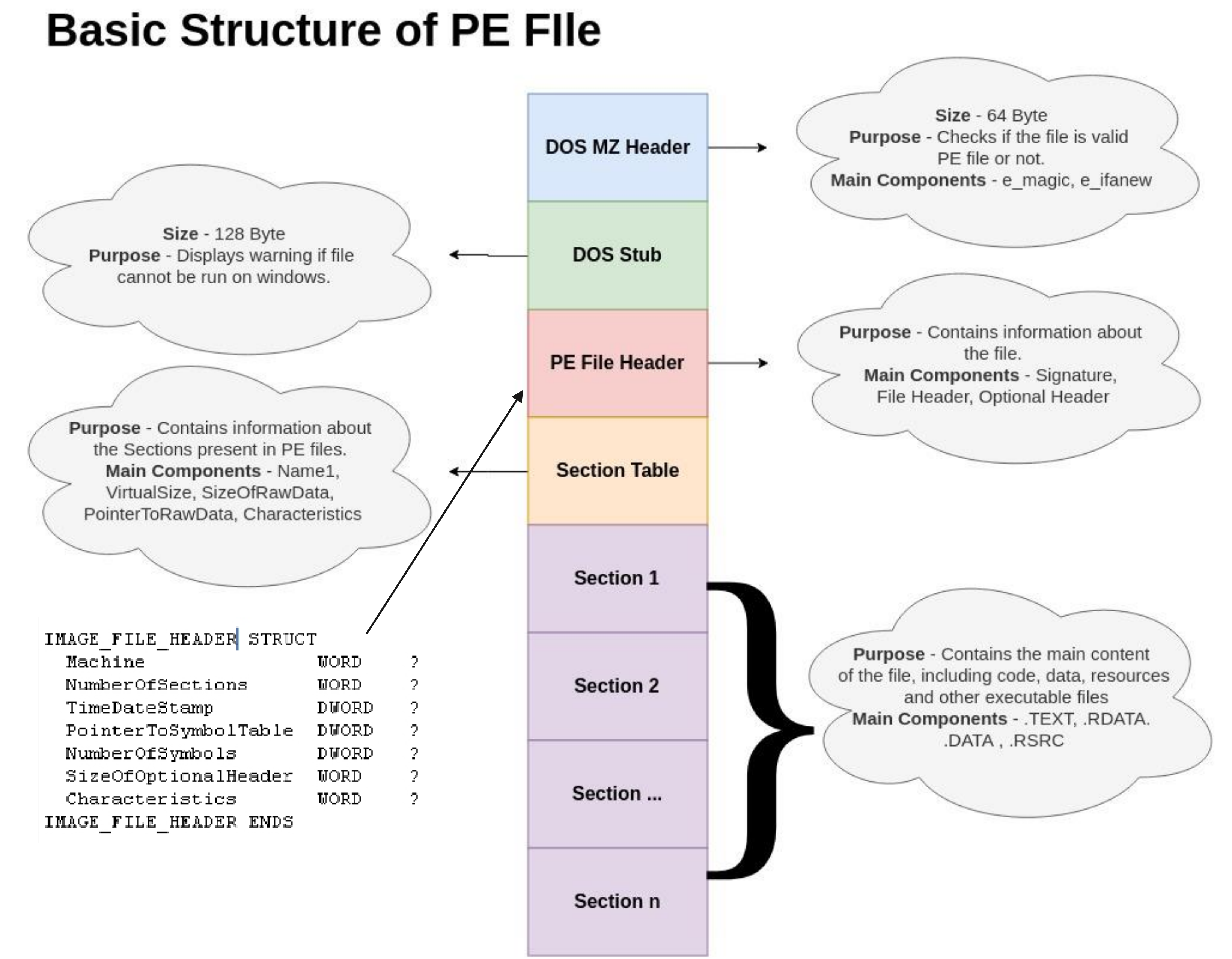
- **Handles -** A handle is process specific numeric identifier which is an index into the processes private handle table.
- **Processes -** A process is really just an isolated memory address space that is used to run a program.
- **Process Initialization -** When a new process calls the **Win32 API CreateProcess**, the API creates a process object and allocates a new memory address space for the process.
- **Threads -** A thread is a data structure that has a **CONTEXT data structure**. At ant given moment, each processor in the system is running one thread.
- **Context Switch -** Context switch is the thread interruption.
- **Win32 API -** An **API** is a set of functions that the operating system makes available to application programs for communicating with the OS.
- **System Calls -** A system call is when a user mode code needs to call a kernel mode function. This occurs when an application calls an operating system API.
- **PE Format -** The Windows executable format - PE (Portable Executable).
- **DLL's -** DLL's allow a program to be broken into more than one executable file.



Structure

PE Header

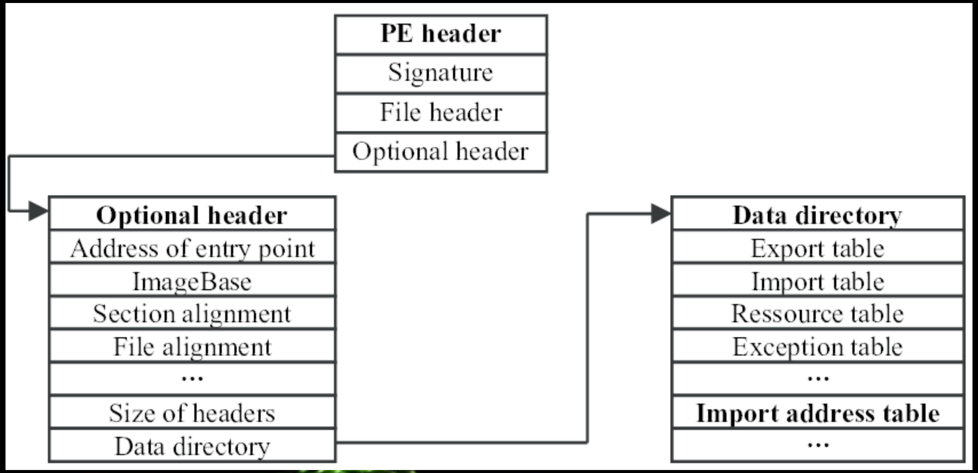
- PE File Header
  - **Number of sections**
  - **Machine** – identifies the type of machine, such as Intel, AMD
  - **Timedatestamp** – represents the time when the linker or compiler produced this file
  - **Characteristics** – the value of the flag helps to identify if the file is a DLL or an executable
- Section Table – contains information about the ***Sections*** present in the PE files.
- PE File Section – main content
  - **.text** – CODE – where all instructions reside. The entry point is located here.
  - **.rdata** – contain the imports and exports information. Contain read-only data used by the program: literals, constant strings, etc.
  - **.data** – contains the “global data” of the program, that can be accessed from anywhere
  - **.rsrc** – contains resources such as images, icons, menu, etc. used by the executable.



Structure

Useful Information

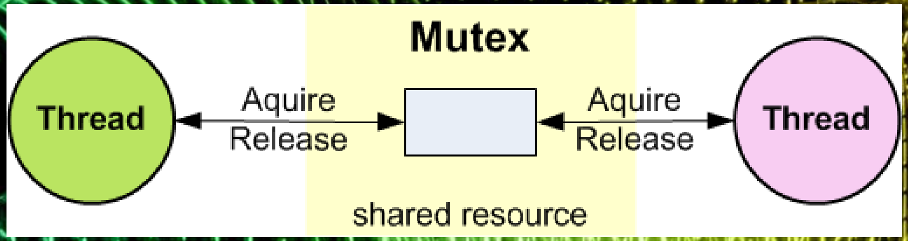
- **Linked Subroutines & Functions**
  - **Dynamic (DLL) vs Static (Inserted into the final executable)**
- **Imports**
  - **Shows the APIs (functions) used by the program/executable that are contained in external libraries**
  - **This can help the analyst to understand and deduce the specimen functionality**
  - **NOTE! Some adversaries add additional functions (not particularly utilized by the malware). It's important to look for API call patterns associated with malware behavior.**
- **Exports**
  - **Stores the names of the APIs functions exported by a DLL and the *relative virtual address (RVA)* where the function can be found.**
  - **Seldomly used in *hooking* activities by adversaries**



Structure

Useful Information

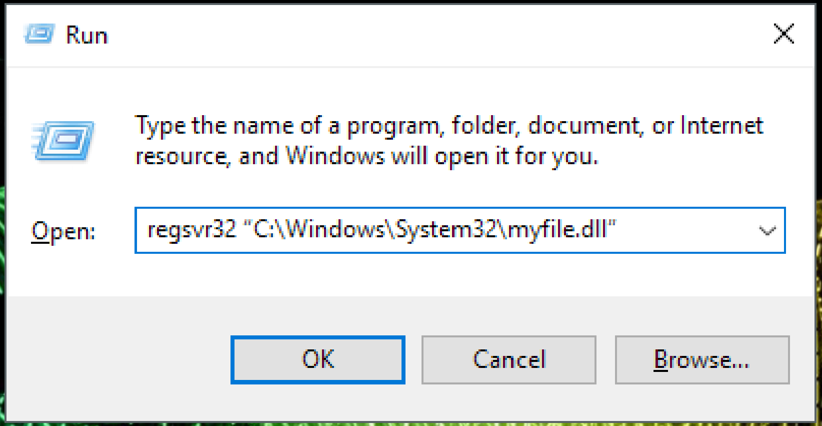
- **Mutex**
  - **Def:** *A mutex object is a synchronization object whose state is set to signaled when it is not owned by any thread, and non signaled when it is owned. Only one thread at a time can own a mutex object*
  - **Example:** *to prevent two threads from writing to shared memory at the same time, each thread waits for ownership of a mutex object before executing the code that accesses the memory. After writing to the shared memory, the thread releases the mutex object.*
  - **Purpose:** Malicious software sometimes uses mutex objects to avoid infecting the system more than once, as well as to coordinate communications among its multiple components on the host.
  - It's relatively uncommon for legitimate programs to use mutex names that are completely random.
  - In some cases, malware might dynamically generate mutex names to attempt to avoid detection. Also, not all malware uses mutex objects.
  - Usage:
    - Use mutexes as Indicators of Compromise to identify potentially infected system
    - In some case, the mutex name (if static) can be used to stop the propagation of malware



Structure

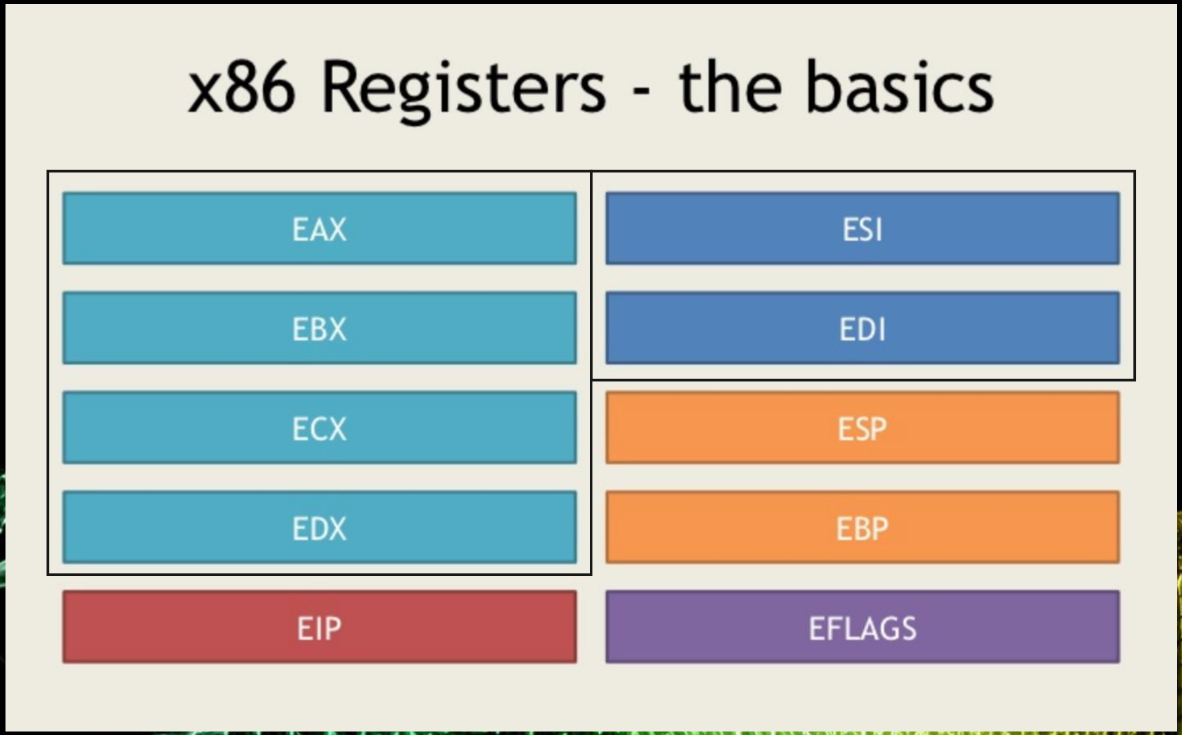
Useful Information

- **Regsvr32**
  - **Def:** *Regsvr32.exe is a command-line program used to register and unregister object linking and embedding controls, including dynamic link libraries (DLLs), on Windows systems.*
  - Regsvr32.exe is also a Microsoft signed binary.
  - Regsvr32 can be used to execute arbitrary binaries.
  - Adversaries may take advantage of this functionality to proxy execution of code to avoid triggering security tools that may not monitor execution of, and modules loaded by, the regsvr32.exe process because of whitelists or false positives from Windows using regsvr32.exe for normal operations.
  - Usage:
    - Can be used to load COM scriptlets to execute DLLs under user permissions
    - Since regsvr32.exe is network and proxy aware, the scripts can be loaded by passing a uniform resource locator (URL) to file on an external Web server as an argument during invocation. This method makes no changes to the Registry as the COM object is not actually registered, only executed (**"Squiblydoo" attack**)
    - Regsvr32.exe can also be leveraged to register a COM Object used to establish Persistence (**Component Object Model Hijacking**)



Assembly Language Primer

- Most of the work done in reverse engineering will be with assembler language
- Topics:
  - **Registers** – Are places in computer memory where data is stored.
  - The Inter 32-bit x86 registers:
    - **EAX** - Extended Accumulator Register
    - **EBX** - Extended Base Register
    - **ECX** - Extended Counter Register
    - **EDX** - Extended Data Register
    - **ESI** - Extended Source Index
    - **EDI** - Extended Destination Index
    - **EBP** - Extended Base Pointer
    - **ESP** - Extended Stack Pointer
    - **EIP** - Extended Instruction Pointer



Assembly Language Primer

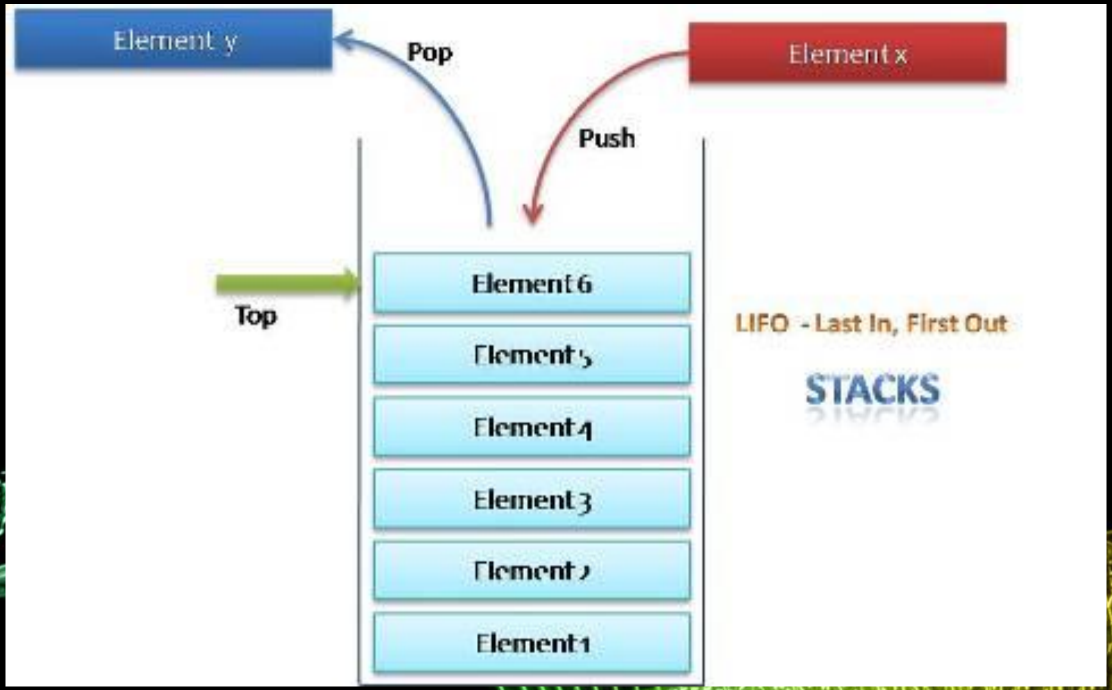
- Segment Registers:
  - Stack Segment (SS). Pointer to the stack.
  - Code Segment (CS). Pointer to the code.
  - Data Segment (DS). Pointer to the data.
  - Extra Segment (ES). Pointer to extra data ('E' stands for 'Extra').
  - F Segment (FS). Pointer to more extra data ('F' comes after 'E').
  - G Segment (GS). Pointer to still more extra data ('G' comes after 'F').

- Flags - Flags are a single bit that indicates status of a register. A flag can only be **SET** or **NOT SET**. Flags:
  - Z – Flag
  - O – Flag
  - C – Flag

Assembly Language Primer

- **Stack** - The stack is a part of memory where you can store different things for late use.
- Instructions - Assembler language has a small number of fundamental commands:

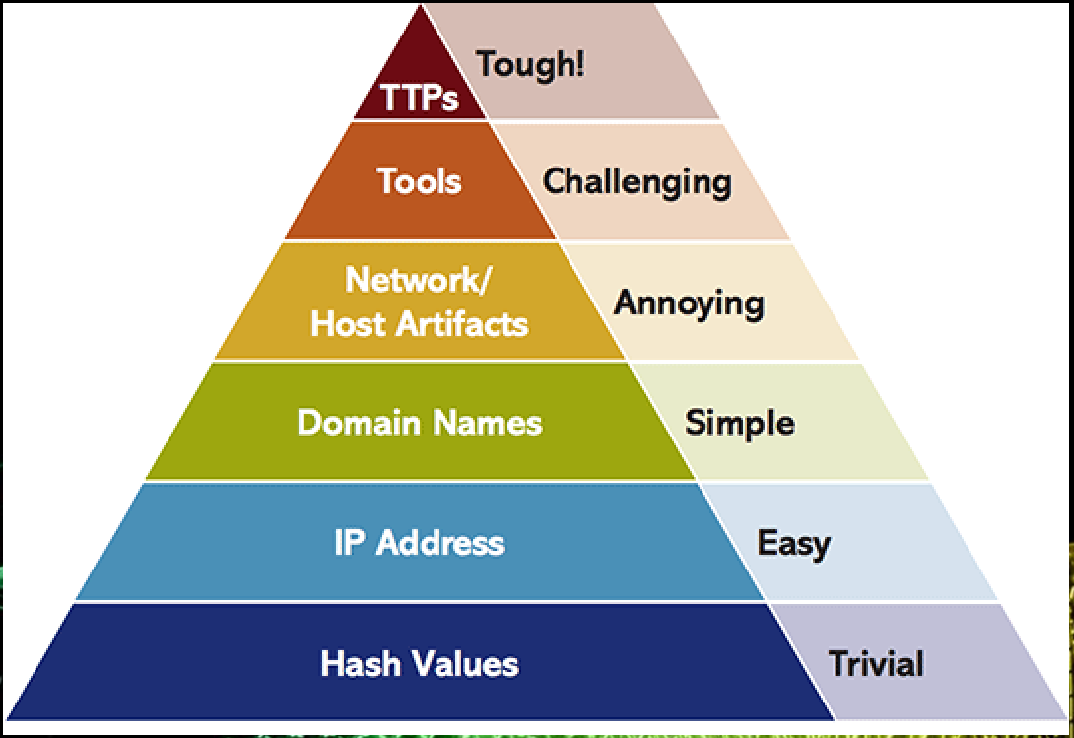
- ADD
- AND
- CALL
- POP
- PUSH
- CMP
- DIV
- IMUL
- JUMP
- MOV
- OR
- XOR
- RET
- SUB
- TEST
- DEC
- INC



Signatures

Preamble – IOCs Pyramid of Pain

- **The pyramid describes two aspects:**
  - **How much “pain” (hence the term in the title) will the blocking of an IOC inflict on the adversary; i.e. how much work will the adversary need to perform in order to bypass the analyst block**
  - **Seen from another perspective, the pyramid shows how easy (and trivial) is to change an indicator, or simply said how “volatile” is an indicator of compromise**
- **We will focus on the second description**



Signatures

Antivirus

- **Hashing – a fingerprint for malware**
  - a common method used to uniquely identify malware
- **Heuristic**
  - behavioral and pattern-matching analysis
- **Strings**
  - A program contains strings if it prints a message, connects to a URL, or copies a file to a specific location.
  - Search for:
    - URL
    - IP Addresses
    - Registry Locations
    - System process names
    - HTTP Methods
    - Etc.

Signatures

Packers & Obfuscated malware

- The original reason packers were used was for compressing the executable:
  - Bandwidth reduction
  - To save disk space
- Adversaries often use packers or obfuscate their malware in order to:
  - Bypass anti-virus detection
  - Prevent Reverse Engineering
- Packed/Obfuscated malware:
  - Always contain few strings, which will hinder the static analysis process
  - Most likely include “*LoadLibrary*” and “*GetProcAddress*” functions, which are used to load and gain access to additional functions
  - Has fewer imports

Packed

RVA	Name
0101AE3Ch	kernel32.dll

RVA	Hint	Name
0101AE00h	0000h	LoadLibraryA
0101AE04h	0000h	GetProcAddress
0101AE08h	0000h	VirtualAlloc
0101AE0Ch	0000h	VirtualFree

Unpacked

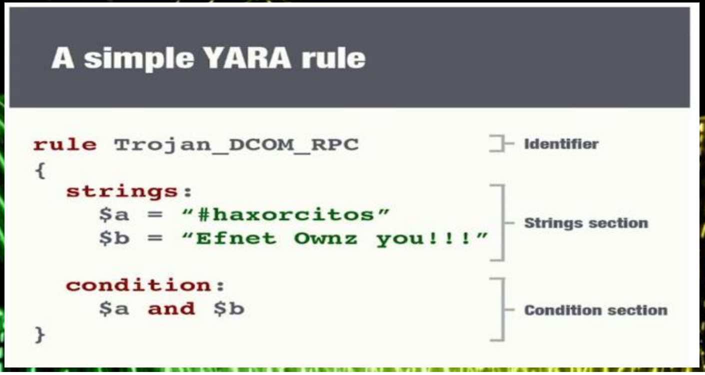
RVA	Name
01007AACh	comdlg32.dll
01007AFAh	SHELL32.dll
01007B3Ah	WINSPOOL.DRV
01007B5Eh	COMCTL32.dll
01007C76h	msvcrt.dll
01007D08h	ADVAPI32.dll
010080ECh	KERNEL32.dll
0100825Eh	GDI32.dll
0100873Ch	USER32.dll

RVA	Hint	Name
010012C4h	000Fh	PageSetupDlgW
010012C8h	0006h	FindTextW
010012CCh	0012h	PrintDlgExW
010012D0h	0003h	ChooseFontW
010012D4h	0008h	GetFileTitleW
010012D8h	000Ah	GetOpenFileNameW
010012DCh	0015h	ReplaceTextW
010012E0h	0004h	CommDlgExtendedError
010012E4h	000Ch	GetSaveFileNameW

Signatures

YARA Framework

- An open source tool aimed to help malware researchers identify and classify malware samples.
- The analyst can create descriptions of malware families based on textual or binary patterns. Each description consists of a set of strings and a Boolean expression that determine its logic
- The framework can be leveraged in al Incident Response phases
  - **Preparation**: in conjunction with CTI, Yara scan engine can accommodate the indicators in it's rules.
  - **Identification**: with YARA an enterprise scan can be performed to identify the infected systems
  - **Containment**: if new indicators are discovered IR team will return to Identification phase and tune the rules and conduct a new scan again
  - **Eradication:** the cleanup phase as well as the blocking of malicious IP addresses, enterprise password changes, this phase does not rely on the YARA scan engine
  - **Recovery:** reestablishment of affected systems back into the organization, this phase does not feature YARA
  - **Lessons Learned**: new revealed indicators can be imported into technologies that features the YARA scan engine



Signatures

Armored Malware

- **Encryption**
- **Compression**
- **Obfuscation**
- **Anti-Patching**
- **Anti-Tracing**
- **Anti-Unpacking**
- **Virtual Env Detection**
- **Polymorphic/Self-Mutating**
- **Password Protected**
- **Configuration Files**

Example – Malware Analysis Framework

- In the following slides we will go through the entire Malware Analysis process

OSINT – Open Source Intelligence

virustotal

SHA256: 1e9f21f514ee4793cfae7baa21549be0d9b432c59513d2efed860c2b1501da39

File name: vdaudio.dll

Detection ratio: 41 / 57

vdaudio.dll

Analysis | File detail | Additional information | Comments 1

Antivirus	Result
ALYac	Gen:Variant.Symmi.60615
AVG	Proxy-Agent.S

PE exports

gewayX
gewayZ
vdaudio

IP Address	Port/Protocol
85.17.114.55 OSINT	53 TCP
217.118.19.171 OSINT	53 TCP
85.25.110.2 OSINT	

Associated Artifacts for 217.118.19.171

Associated SHA256
c8b28705fc1b25208462ae9913e6c4821f537e276eee4f98a51035bc610402eb
911031e9e8e27368e02bcead8121a13ff526b317304d6836579689ee43e3a53a
4813700e5a48e38c3b2101122995b7d95f3078195400d1a976c9d5bc4fd91d0a84
7b8511d61295dba4cc4c8130895ab88337cc2eb26b0cbe038dd293dcdb71e188

PAYLOAD SECURITY

Domain	Address	Country
cn.mnemonicarx.biz	220.74.146.251	Korea Republic of
cm.mnemonicarx.biz	80.37.239.187	Spain

Creates mutants

details "\Sessions\1\BaseNamedObjects\LXCVOIMGIXSORTA1"
source Created Mutant
relevance 3/10
research Show me all reports matching the same indicator

PASSIVE TOTAL

cn.mnemonicarx.biz

OPEN THREAT EXCHANGE

Resolve	Location	Source	Tags
220.74.146.251	KR	emerging_threats, riskiq, pingly, virustotal, kaspersky	Kixs-As-Krkorea Telecom

Associated Files dceb91a3aace0c732f5732584fe7eac2635546f10df2bd0ce0330a9d3730016d

Mutexes
LXCV0IMGIXS0RTA1
TXA19EQZP13A6JTR

IDS Signatures ETPRO TROJAN Bunitu Covert ... 85.17.114.55

Dropped Files llmedia.dll 4cd0d519f0c97f7d6b4030cb219

Microsoft
Malware Protection
Win32/Bunitu

Static Properties Analysis

BinText

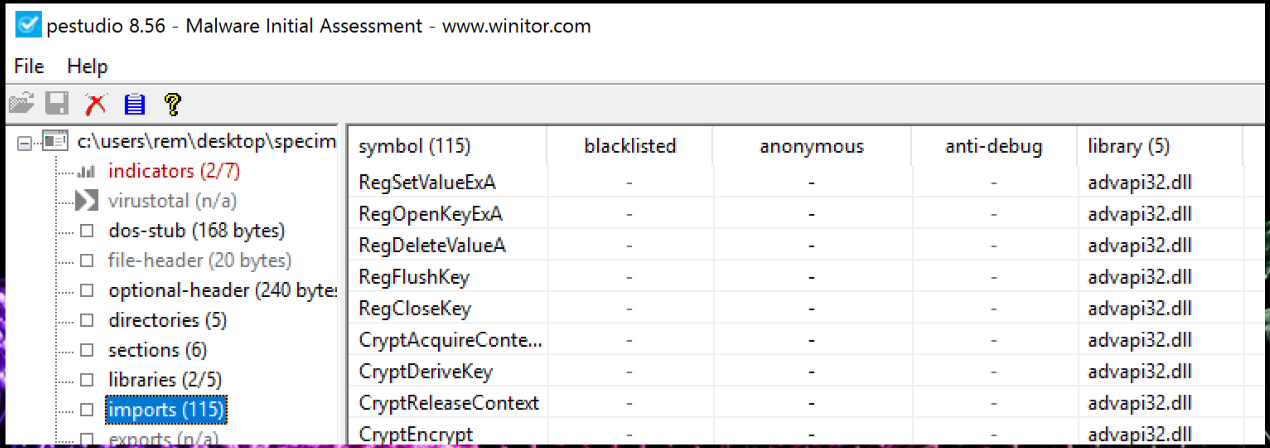
- By examining the embedded strings we begin to formulate theories
- IOCs for detection:
  - File system: brbconfig.tmp
  - Network: %s?i=%s&c=%s&p=%s
- Persistence mechanism:
  - Registry: …\Windows\CurrentVersion\Run
- Communication mechanism:
  - Network: HTTP/1.1
  - User-Agent: Mozilla/4.0…

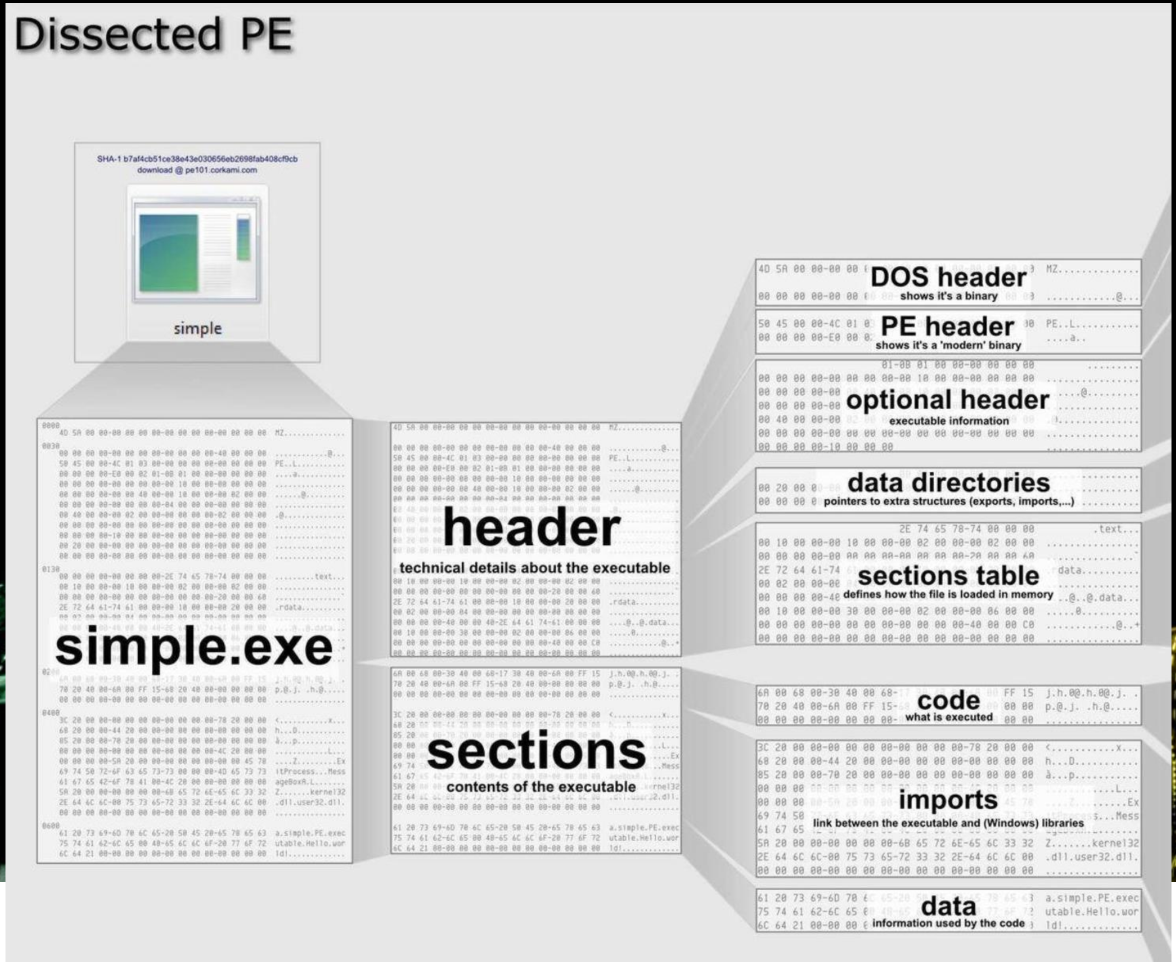
A	00000000E560	00000000E4ED	0	GetProcessWindowStation
A	00000000E578	00000000E505	0	GetUserObjectInformationW
A	00000000E598	00000000E525	0	GetLastActivePopup
A	00000000E5B0	00000000E53D	0	GetActiveWindow
A	00000000E5C0	00000000E54D	0	MessageBoxW
A	00000000E5F8	00000000E585	0	CONFIG
A	00000000E600	00000000E58D	0	brbconfig.tmp
A	00000000E610	00000000E59D	0	YnJiYm90
A	00000000E640	00000000E5CD	0	sleep
A	00000000E648	00000000E5D5	0	encode
A	00000000E660	00000000E5ED	0	%s?i=%s&c=%s&p=%s
A	00000000E678	00000000E605	0	APPDATA
A	00000000E680	00000000E60D	0	Software\Microsoft\Windows\CurrentVersion\Run
A	00000000E6B0	00000000E63D	0	brbbot
A	00000000E730	00000000E6BD	0	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0)
A	00000000E770	00000000E6FD	0	HTTP/1.1
A	00000000E780	00000000E70D	0	Connection: close

Static Properties Analysis

PeStudio

- A static analysis tool that, amongst many other features, has an :Indicator” section that highlights potential malicious aspects.
- Shows information about the file sections and imports
- It also shows the imports

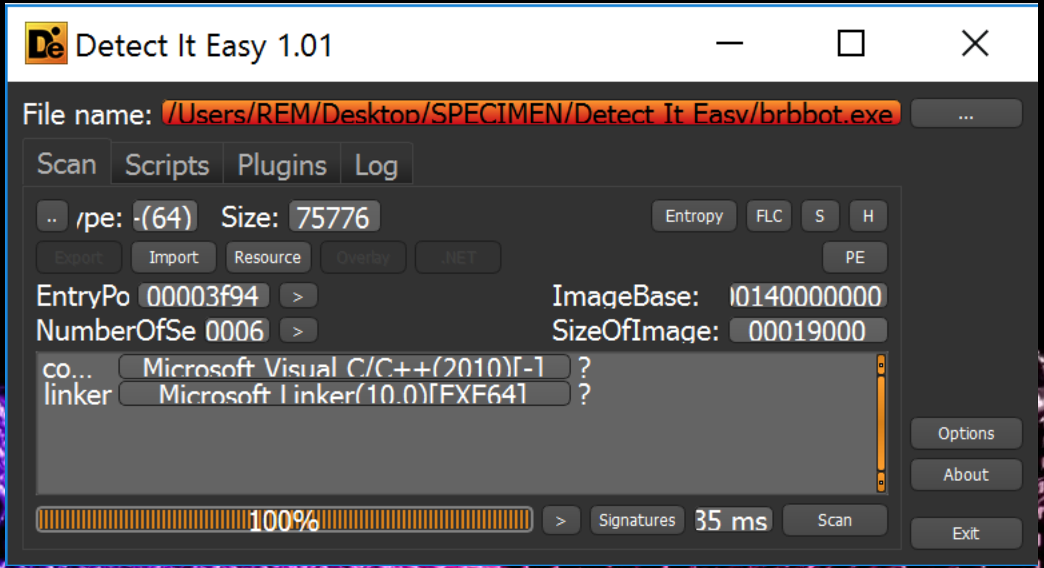


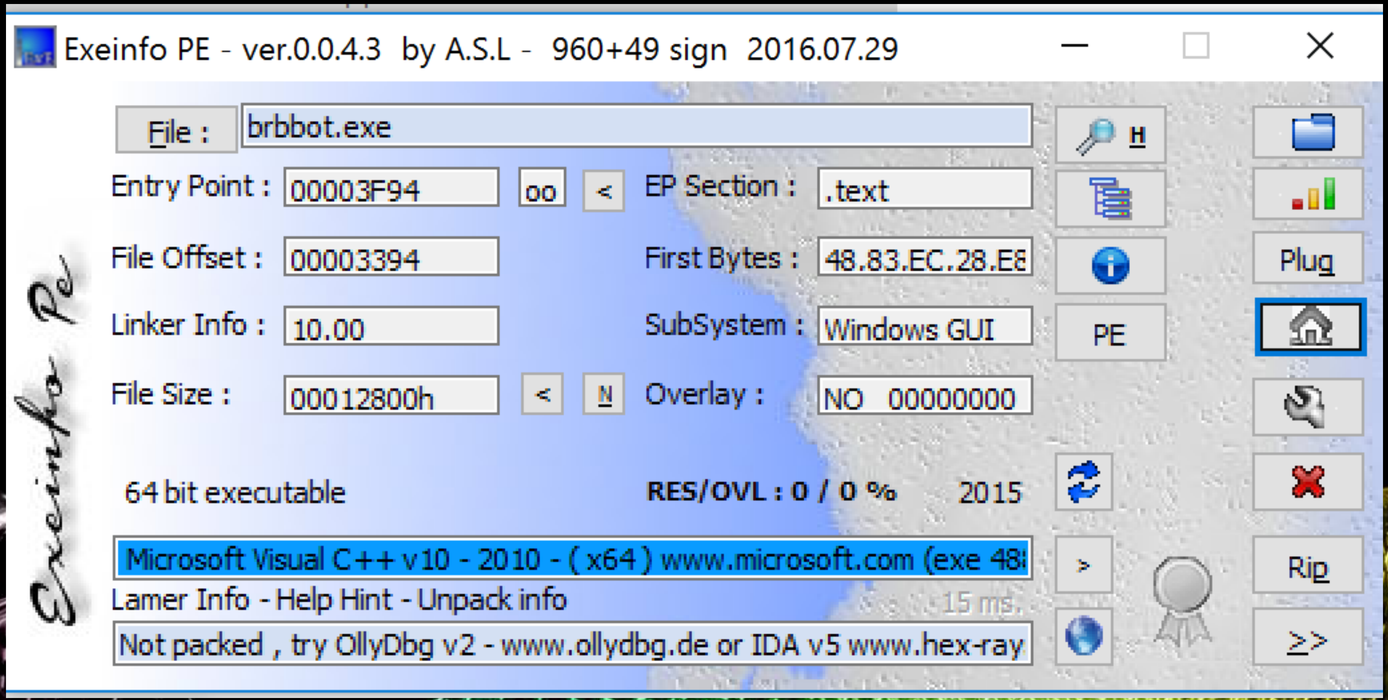


Static Properties Analysis

Detect It Easy & Exeinfo PE

- Show PE header details and try to recognize the tools that created the files (programming langue and packers)





Static Properties Analysis

Results

- You discovered potential IOCs
- You obtained hashes and other details useful for researching the sample on the web
- You identified indicators of potential functionality based on references to Windows APIs
- You formulated theories that you might validate using further analysis steps

BinText

PeStudio

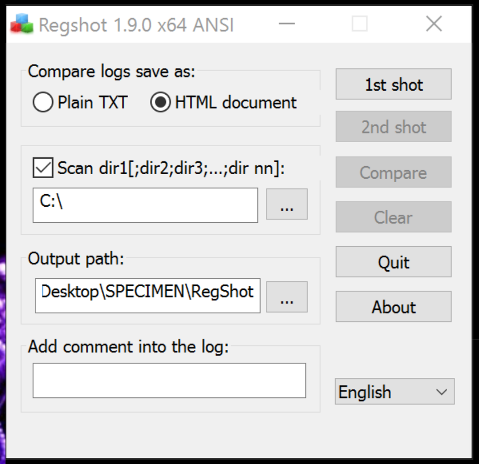
Detect It Easy

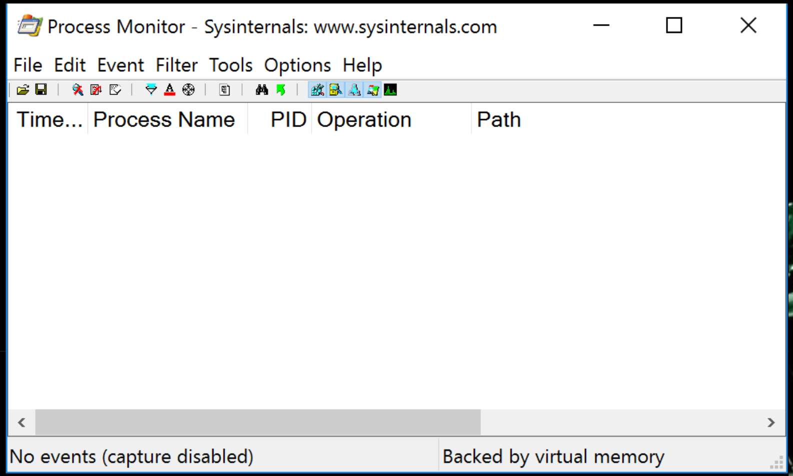
Exeinfo PE

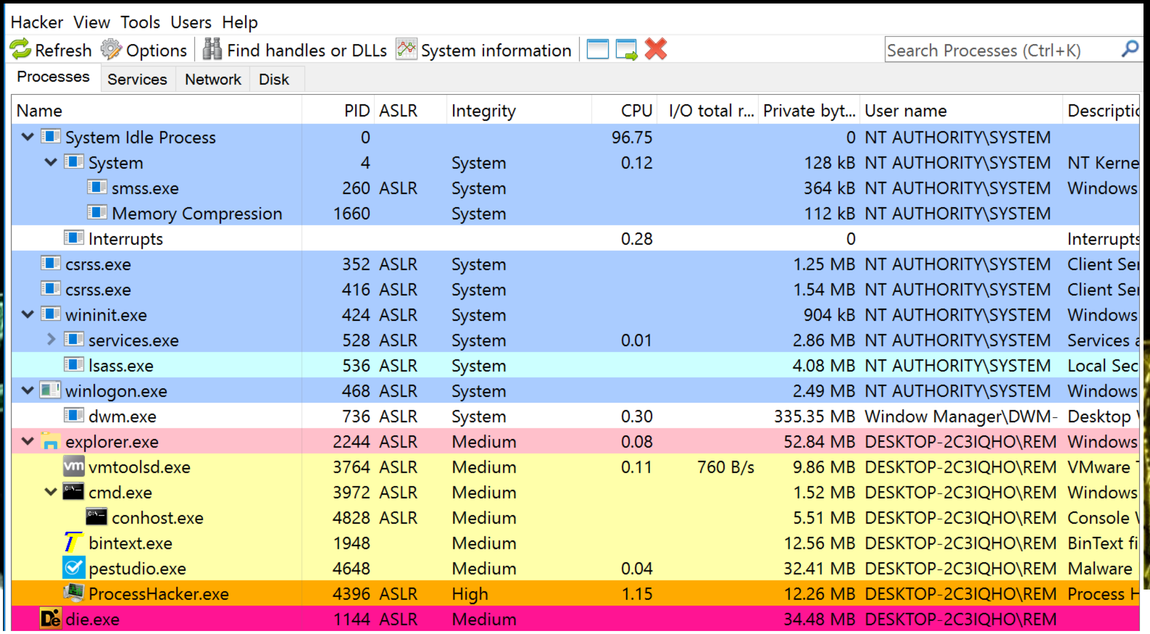
Behavioral Analysis

Capture the pre-detonation state

- Before you infect the machine in order to observer the behavior, you need to capture the current state (clean/un-infected)
- We will focus on three applications that cover different aspect of the OS:
  - **Process Hacker** – like the built-in Task Manager, similar to Process Explorer
  - **Process Monitor** – records interactions of processes with registry, file system, network and other processes
  - **Regshot** – highlights changes to the file system and registry
  - **Wireshark** – sniffs the network and captures packets







Behavioral Analysis

Detonation

- Before detonating the specimen, capture a registry image with **Regshot**
- Open **Process Hacker**
- Open **Process Monitor** and start monitoring
- Open **Wireshark** and start capturing
- **Detonate** the specimen by running the process
- Observe the malicious process in **Process Hacker** and after 1 minute of running terminate the process.
- Stop monitoring in **Process Monitor**, and stop capturing in **Wireshark**
- Take a second registry image with **Regshot** for comparison

Behavioral Analysis

Analyzing the results - Regshot

- Let’s start with the **Regshot** comparison. We observe that:
  - One of the **Values Added in the registry** is a key in order to start the specimen at each system reboot – **PERSISTENCE.**
  - One of the added files is the one found in the **Static Analysis** phase – **SECOND/NEXT STAGE of the infection process**

Files added: 7

C:\Users\REM\AppData\Local\Temp\wireshark_622D289E-
C:\Users\REM\AppData\Roaming\Microsoft\Crypto\RSA\S-1
C:\Users\REM\AppData\Roaming\Microsoft\Protect\S-1-5-2
C:\Users\REM\AppData\Roaming\brbconfig.tmp

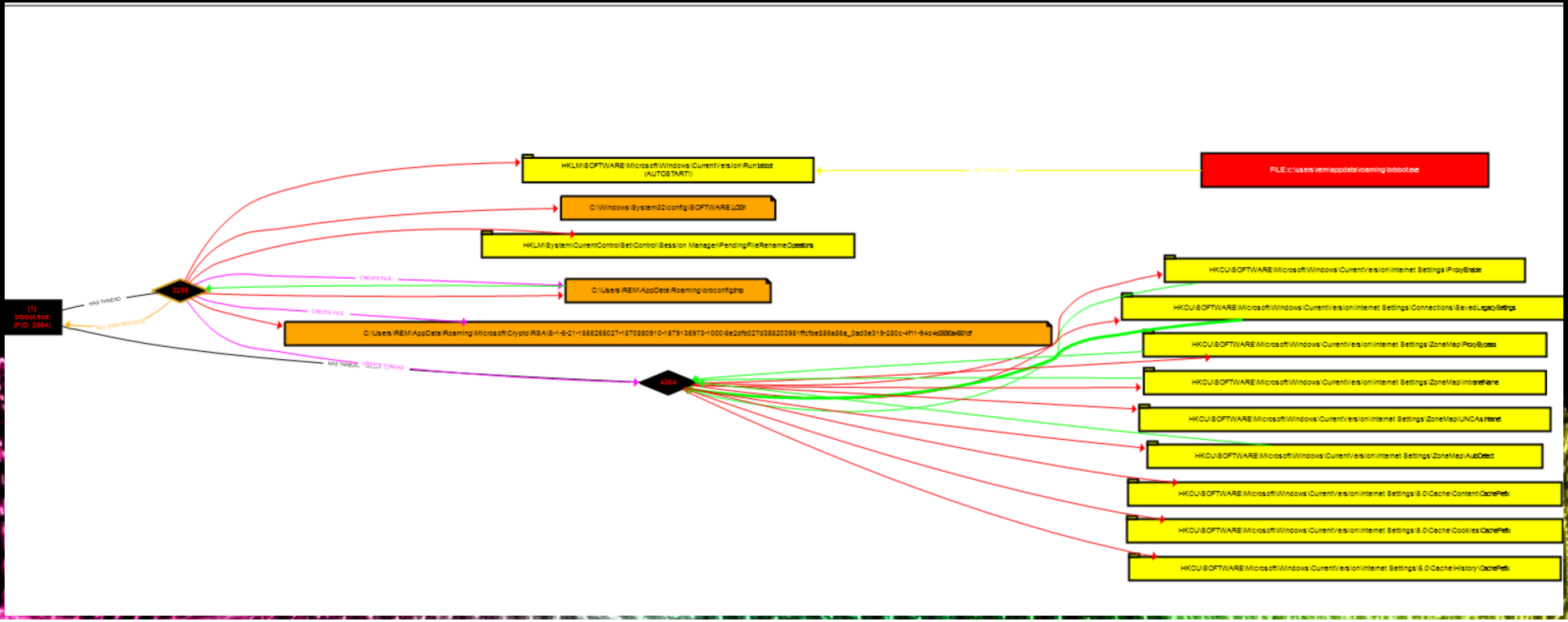
Values added: 68

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\brbbot: "C:\Users\REM\AppData\Roaming\brbbot.exe"
HKLM\SYSTEM\ControlSet001\Control\Session Manager\PendingFileRenameOperations: 5C 3F 3F 5C 43 3A 5C 55 7
HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\PendingFileRenameOperations: 5C 3F 3F 5C 43 3A 5C
HKU\S-1-5-21-1866265027-1870850910-1579135973-1000\SOFTWARE\Microsoft\Windows\CurrentVersion\Explore

Behavioral Analysis

Analyzing the results – Process Monitor

- It can be difficult to filter the output for Process Monitor. In order to analyze the results we will use **ProcDOT**
- **ProcDOT** generates a map of all the activities performed by the malicious sample



Behavioral Analysis

Wireshark

- We notice that **Wireshark** shows logs of a suspicious DNS query from the infected system
- Because we haven’t enabled DNS service in the lab, the response to this query is an **ICMP Destination unreachable**, indicating that the system cannot process this connection attempt
- We can utilize a second VM in and enable **fakedns** on it in order to resolve DNS requests

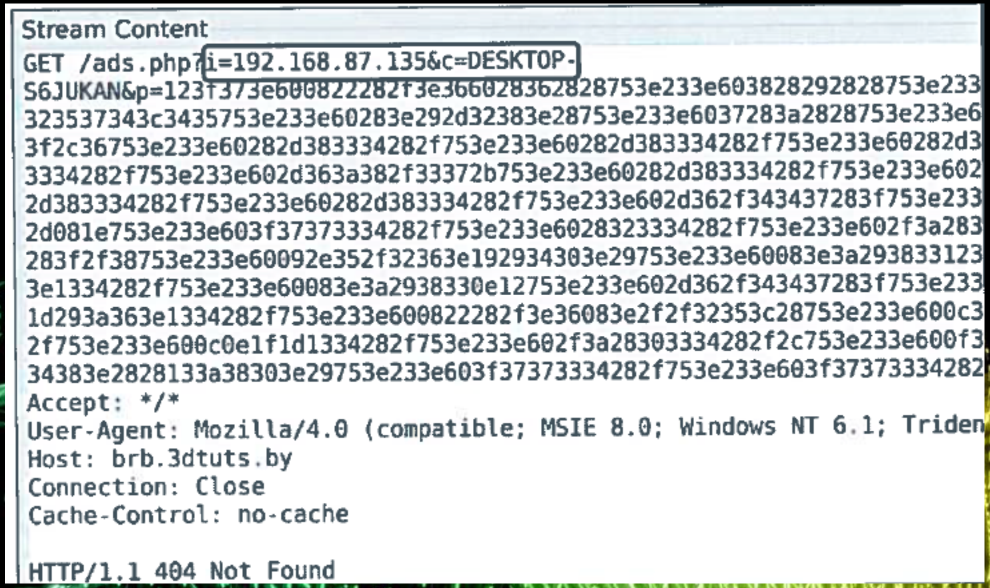
Apply a display filter … <Ctrl-/> Expressio

No.	Time	Source	Destination	Protocol	Length	Info
257	703.733525	Vmware_c0:00:01	Vmware_ba:9c:13	ARP	60	192.168.209.1 is at 00:50:56:c0:00:01
258	718.393432	192.168.209.129	192.168.209.1	DNS	76	Standard query 0x01f0 A fs.microsoft.com
259	718.393708	192.168.209.1	192.168.209.129	ICMP	70	Destination unreachable (Port unreachable)
260	718.393778	192.168.209.129	192.168.209.1	DNS	76	Standard query 0x01f0 A fs.microsoft.com
261	718.393883	192.168.209.1	192.168.209.129	ICMP	70	Destination unreachable (Port unreachable)
262	718.393960	192.168.209.129	192.168.209.1	DNS	76	Standard query 0x01f0 A fs.microsoft.com
263	718.394092	192.168.209.1	192.168.209.129	ICMP	70	Destination unreachable (Port unreachable)
264	718.394148	192.168.209.129	192.168.209.1	DNS	76	Standard query 0x01f0 A fs.microsoft.com
265	718.394270	192.168.209.1	192.168.209.129	ICMP	70	Destination unreachable (Port unreachable)
266	718.394337	192.168.209.129	192.168.209.1	DNS	76	Standard query 0x01f0 A fs.microsoft.com
267	718.394433	192.168.209.1	192.168.209.129	ICMP	70	Destination unreachable (Port unreachable)
268	778.283673	192.168.209.129	192.168.209.255	BROWSER	258	Domain/Workgroup Announcement WORKGROUP, NT Workstati…

Behavioral Analysis

Wireshark – with DNS resolution capabilities

- After we enable the DNS service, we reinfect the machine while capturing the traffic with **Wireshark**
- We notice that the specimen attempted to make an HTTP connection
- The HTTP **GET Request** is an attempt to submit data. This can be recognized by the ***"&"*** sign present in the request
- The "p" parameter observed in the **TCP Stream** tells us that the string is encoded in hexadecimal
- When we attempt to convert the string to ASCII, we discover that the output isn't simple ASCII encoded as hex, and might be encrypted

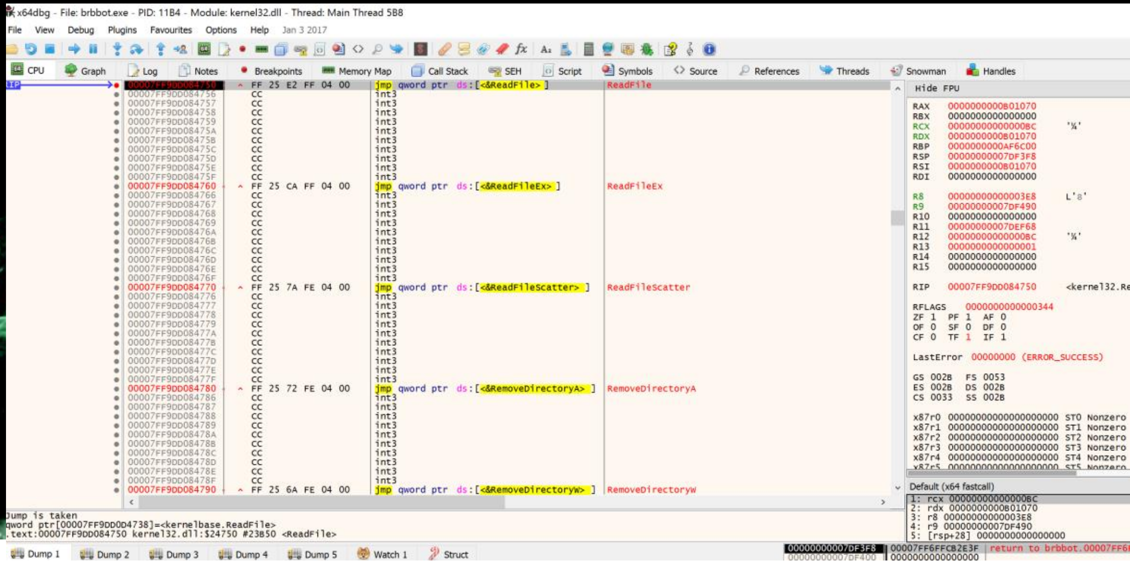

Stream Content

```
GET /ads.php?i=192.168.87.135&c=DESKTOP-
S6JUKAN&p=123f373e600822282f3e36602836282875 3e233e60382829282875 3e233
323537343c3435753e233e60283e292d32383e28753e233e6037283a2828753e233e6
3f2c36753e233e60282d383334282f753e233e60282d383334282f753e233e60282d3
3334282f753e233e602d363a382f33372b753e233e60282d383334282f753e233e602
2d383334282f753e233e60282d383334282f753e233e602d362f343437283f753e233
2d081e753e233e603f37373334282f753e233e602832333 4282f753e233e602f3a283
283f2f38753e233e60092e352f32363e192934303e29753e233e60083e3a29383312 3
3e1334282f753e233e60083e3a2938330e12753e233e602d362f343437283f753e233
1d293a363e1334282f753e233e600822282f3e36083e2f2f32353c28753e233e600c3
2f753e233e600c0e1f1d1334282f753e233e602f3a2830333 4282f2c753e233e600f3
34383e2828133a38303e29753e233e603f37373334282f753e233e603f37373334282
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Triden
Host: brb.3dtuts.by
Connection: Close
Cache-Control: no-cache

HTTP/1.1 404 Not Found
```

Code Analysis

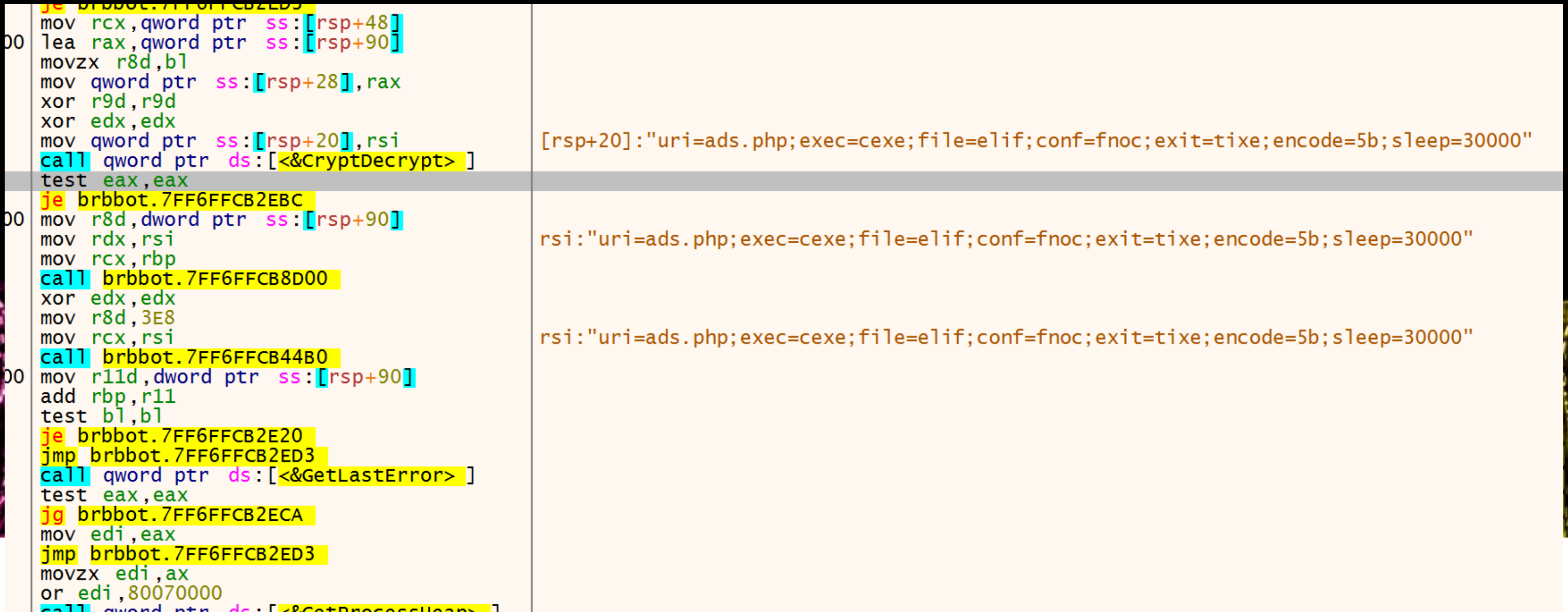
- Behavioral analysis provided us a lot results, but there are still questions to be answered, such as:
  - What does the **tmp** file contain?
  - What was being sent out in the encoded parameter?
  - If and how the attacker controls the specimen?
- In order to answer the above questions we need to perform **Code Analysis**
- The first step is to try and view what does the specimen uses in order to read the **tmp** file and does it decode it.
  - We observed in **PeStudio**, the “**ReadFile**” function



- We load the file in a debugger (x64dbg) and search for the above-mentioned function
  - We set a breakpoint with the **SetBPX ReadFile** command, and afterwards **RUN** the specimen
  - The function expects the pointer to the file as a first parameter ***hFile***
  - By examining the **handles** contained in the RCX register we identify the value **BC**

Code Analysis

- In the **Handles tab** we search for the mentioned address we identify the **tmp** file
- The next step is to allow the code to run in order to reach the code that called it
  - Debug -> Run to user code
- We observe that there is a function called: “CryptDecrypt”.
  - In order to verify this aspect, we select the instruction <u>**test eax, eax**</u> which comes immediately after the above-mentioned function and Debug -> Run until selection
  - In the **Stack** region we see the output of CryptDecrypt
- We observe the following:
  - A list of possible C2 commands used to control the specimen
  - A web page to retrieve
  - A possible polling timer
  - Includes an encoding key (5b)



Code Analysis

Decode the captured traffic

- By utilizing the discovered key (5b) we try to decode the string by performing a XOR operation
- We discover that the submitted data contains telemetry about the infected system, more specifically a list of executables

```
Stream Content
GET /ads.php?i=192.168.87.135&c=DESKTOP-
S6JUKAN&p=123f373e600822282f3e3660283628 28753e233e60382829282 8753e233
323537343c3435753e233e60283e292d32383e28753e233e6037283a2828753e233e6
3f2c36753e233e60282d383334282f753e233e60282d383334282f753e233e60282d3
3334282f753e233e602d363a382f33372b753e233e60282d383334282f753e233e602
2d383334282f753e233e60282d383334282f753e233e602d362f343437283f753e233
2d081e753e233e603f37373334282f753e233e602832333428 2f753e233e602f3a283
283f2f38753e233e60092e352f32363e192934303e29753e233e60083e3a29383312 3
3e1334282f753e233e60083e3a2938330e12753e233e602d362f343437283f753e233
1d293a363e1334282f753e233e600822282f3e36083e2f2f32353c28753e233e600c3
2f753e233e600c0e1f1d1334282f753e233e602f3a28303334282f2c753e233e600f3
34383e2828133a38303e29753e233e603f37373334282f753e233e603f3737333428 2
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Triden
Host: brb.3dtuts.by
Connection: Close
Cache-Control: no-cache

HTTP/1.1 404 Not Found
```

```
remnux@remnux:~$ xxd -r -p encoded.hex > encoded.raw
remnux@remnux:~$ translate.py encoded.raw decoded.txt 'byte ^ 0x5b'
remnux@remnux:~$ cat decoded.txt
Idle;System;smss.exe;csrss.exe;wininit.exe;csrss.exe;winlogon.exe;services.exe;lsass.exe;
svchost.exe;svchost.exe;dwm.exe;svchost.exe;svchost.exe;svchost.exe;svchost.exe;svchost.e
xe;vmacthlp.exe;svchost.exe;svchost.exe;spoolsv.exe;svchost.exe;svchost.exe;vmtoolsd.exe;
VGAuthService.exe;WmiPrvSE.exe;dllhost.exe;sihost.exe;taskhostw.exe;explorer.exe;msdtc.ex
e;RuntimeBroker.exe;SearchIndexer.exe;ShellExperienceHost.exe;SearchUI.exe;vmtoolsd.exe;s
vchost.exe;ApplicationFrameHost.exe;SystemSettings.exe;WinStore.Mobile.exe;dllhost.exe;ta
skhostw.exe;WUDFHost.exe;TabTip.exe;TabTip32.exe;TrustedInstaller.exe;TiWorker.exe;Proces
sHacker.exe;Procmon.exe;Procmon64.exe;WmiPrvSE.exe;Regshot-x64-ANSI.exe;dllhost.exe;dllho
```

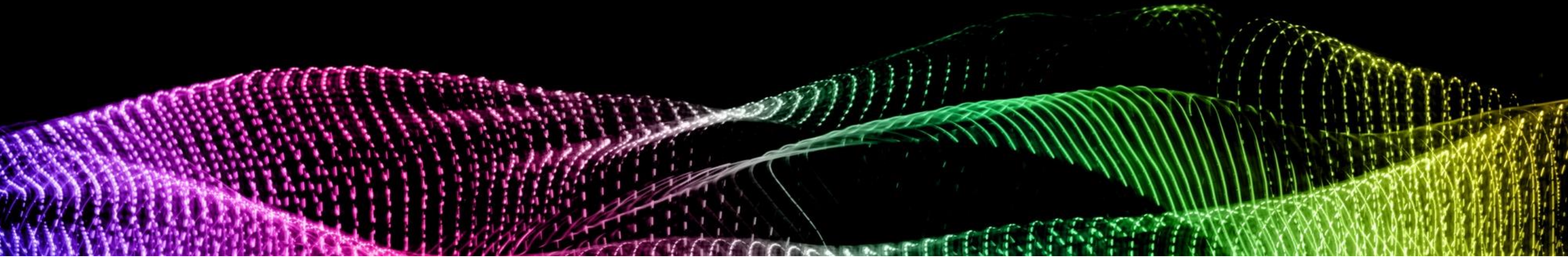
Radare2

Reverse Engineering Framework similar to IDA pro or Ghidra

- r2 is a rewrite from scratch of radare in order to provide a set of libraries and tools to work with binary files

Open-source on github https://github.com/radareorg/radare2

Cutter is a Qt and C++ GUI for radare2



Radare2

Command-line Options

This is an excerpt from the usage help message:

```
$ radare2 -h
Usage: r2 [-ACdfLMnNqStuvwzX] [-P patch] [-p prj] [-a arch] [-b bits] [-i file]
          [-s addr] [-B baddr] [-m maddr] [-c cmd] [-e k=v] file|pid|-|--|=
 --              run radare2 without opening any file
 -               same as 'r2 malloc://512'
 =               read file from stdin (use -i and -c to run cmds)
 -=              perform !=! command to run all commands remotely
 -0              print \x00 after init and every command
 -2              close stderr file descriptor (silent warning messages)
 -a [arch]       set asm.arch
 -A              run 'aaa' command to analyze all referenced code
 -b [bits]       set asm.bits
 -B [baddr]      set base address for PIE binaries
 -c 'cmd..'      execute radare command
 -C              file is host:port (alias for -c+=http://%s/cmd/)
 -d              debug the executable 'file' or running process 'pid'
 -D [backend]    enable debug mode (e cfg.debug=true)
 -e k=v          evaluate config var
 -f              block size = file size
 -F [binplug]    force to use that rbin plugin
 -h, -hh         show help message, -hh for long
 -H ([var])      display variable
 -i [file]       run script file
 -I [file]       run script file before the file is opened
 -k [OS/kern]    set asm.os (linux, macos, w32, netbsd, ...)
 -l [lib]        load plugin file
 -L              list supported IO plugins
```

Opening a file

Open a file in write mode without parsing the file format headers: r2 -nw <file>

E.g. -> For automatic analysis use: “aaa”

See ~? for help.

The ~ character enables internal grep-like function used to filter output of any command

```
root@kali:~/Machines/CTFS/Volga/pwn# r2 notepad
 -- The door can see into your soul.
[0x000009f0]>
```

```
[0x000009f0]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x000009f0]>
```

Looking for interesting things

Radare2 – yara scanning utility

```
[0x00405b42]> yara add ip.r
[0x00405b42]> o ../../../../Malware_samples/WinEXE/pe3packed.exe
[0x00405b42]> yara scan
SEH_Save
SEH_Init
vmdetect
win_mutex
win_files_operation
domain
IsPE32
IsWindowsGUI
IsPacked
HasRichSignature
VMWare_Detection
[0x00405b42]>
```

Looking for interesting things

Pxw command to scan the memory (check px?)

Pd for dezasambling (check p?)

Ax for xrefernce listing from binaries:

```
[0x00405b42]> pd 2 @ 0x40381f
          0x0040381f      6830c04000      push 0x40c030
          0x00403824      e8f70c0000      call fcn.00404520
```

```
[0x00405b42]> ax~section..data
          fcn.004016a0+8575 0x40381f ->      DATA -> 0x40c030 section..data+48
            fcn.00405d4e+18 0x405d60 ->      DATA -> 0x40c00c section..data+12
            fcn.00405d4e+23 0x405d65 ->      DATA -> 0x40c018 section..data+24
            fcn.00405d4e+69 0x405d93 ->      DATA -> 0x40c000 section..data
            fcn.00405d4e+76 0x405d9a ->      DATA -> 0x40c008 section..data+8
           fcn.00405db8+103 0x405e1f ->      DATA -> 0x40c01c section..data+28
           fcn.00405db8+108 0x405e24 ->      DATA -> 0x40c020 section..data+32
           fcn.00405db8+134 0x405e3e ->      DATA -> 0x40c024 section..data+36
           fcn.00405db8+139 0x405e43 ->      DATA -> 0x40c028 section..data+40
```

Also..decompiler?

Ghidra C-decompiler present (to install it -> r2pm -i r2ghidra-dec)

```
[0x00405b42]> s fcn.00404520
[0x00404520]> pdg

// WARNING: [r2ghidra] Failed to match type HANDLE for variable hObject to Decompiler type: Unknown type identifier
// HANDLE
// WARNING: [r2ghidra] Detected overlap for variable var_9h
// WARNING: [r2ghidra] Failed to match type LPCSTR for variable lpFileName to Decompiler type: Unknown type identifier
// LPCSTR
// WARNING: [r2ghidra] Failed to match type LPCVOID for variable lpBuffer to Decompiler type: Unknown type identifier
// LPCVOID
// WARNING: [r2ghidra] Detected overlap for variable lpBuffer
// WARNING: [r2ghidra] Failed to match type LPDWORD for variable lpNumberOfBytesWritten to Decompiler type: Unknown type
// identifier LPDWORD
// WARNING: [r2ghidra] Var arg_4h is stack pointer based, which is not supported for decompilation.

uint32_t __cdecl fcn.00404520(int32_t arg_8h, uint32_t arg_ch, undefined4 lpFileName, int32_t arg_14h)
{
    int32_t iVar1;
    undefined4 uVar2;
    uint32_t uVar3;
    uint32_t var_1ch;
    char lpBuffer;
    int32_t var_14h;
    int32_t lpNumberOfBytesWritten;
    uint8_t var_9h;
    undefined4 hObject;
    int32_t var_4h;

    var_14h = 0;
    var_9h = 0;
    iVar1 = (*_sym.imp.KERNEL32.dll_CreateFileA)(lpFileName, 0x40000000, 2, 0, 4, 0x80, 0);
    uVar2 = (*_sym.imp.KERNEL32.dll_GetTickCount)();
    fcn.00405af0(uVar2);
    if (iVar1 != -1) {
        var_1ch = 0;
        while (var_1ch < arg_ch) {
            lpBuffer = (*(char *)(arg_8h + var_1ch) - (char)arg_14h) - (char)var_1ch;
            if ((0x2f0 < var_1ch) && (var_1ch < 0x2ff)) {
```

Debugging posible

Radare2 gebug option: (r2 -d <file>)

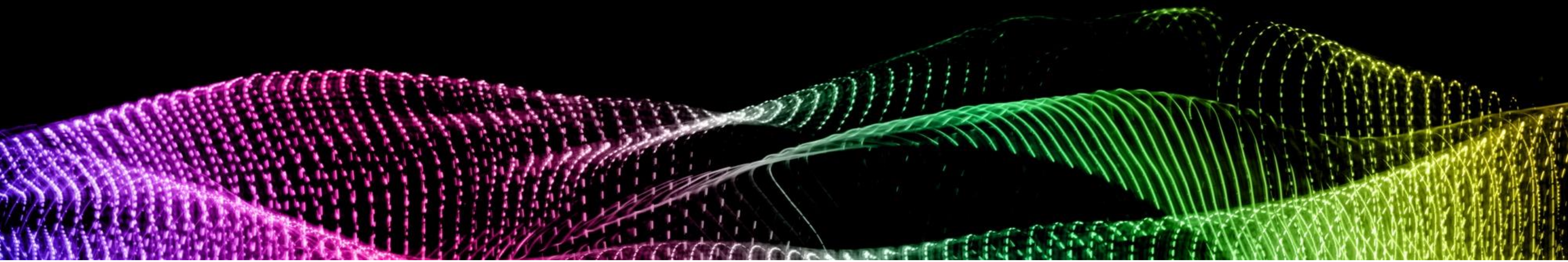
Auto-analisys function (aaa)

Db - set breakpoints(see db?)

Dc - continue(see dc?)

Ds - step (see ds?)

Dm - list memory-maps (see dm?)



Finding virtualalloc, virtualprotect

```
[0x7ffc81f0a250]> afl
0x0040da51    27 49290 -> 924   entry0
[0x7ffc81f0a250]> db entry0
r_w32_dbg_modules/CreateToolhelp32Snaps
```

dc(continue)

```
[0x77d2f0f7]> dc
(2480) loading library at 00000000754D0000 (C:\Windows\SysWOW64\imm32.dll) imm32.dll
hit breakpoint at: 40da51
```

```
[0x0040da51]> dmi KERNELBASE.dll~VirtualAlloc
1774   0x0019d270 0x771cde70 GLOBAL FUNC 0    KERNELBASE.dll                         VirtualAlloc2
1775   0x0019d2f0 0x771cdef0 GLOBAL FUNC 0    KERNELBASE.dll                         VirtualAlloc2FromApp
1776   0x0011c2b0 0x7714ceb0 GLOBAL FUNC 0    KERNELBASE.dll                         VirtualAlloc
1777   0x000f9900 0x7712a500 GLOBAL FUNC 0    KERNELBASE.dll                         VirtualAllocEx
1778   0x00117e70 0x77148a70 GLOBAL FUNC 0    KERNELBASE.dll                         VirtualAllocExNuma
1779   0x0019d320 0x771cdf20 GLOBAL FUNC 0    KERNELBASE.dll                         VirtualAllocFromApp
[0x0040da51]>
```

```
[0x0040da51]> dmi KERNELBASE.dll~VirtualProtect
1783   0x0011b570 0x7714c170 GLOBAL FUNC 0    KERNELBASE.dll                         VirtualProtect
1784   0x0019d360 0x771cdf60 GLOBAL FUNC 0    KERNELBASE.dll                         VirtualProtectEx
1785   0x0019d3e0 0x771cdfe0 GLOBAL FUNC 0    KERNELBASE.dll                         VirtualProtectFromApp
```

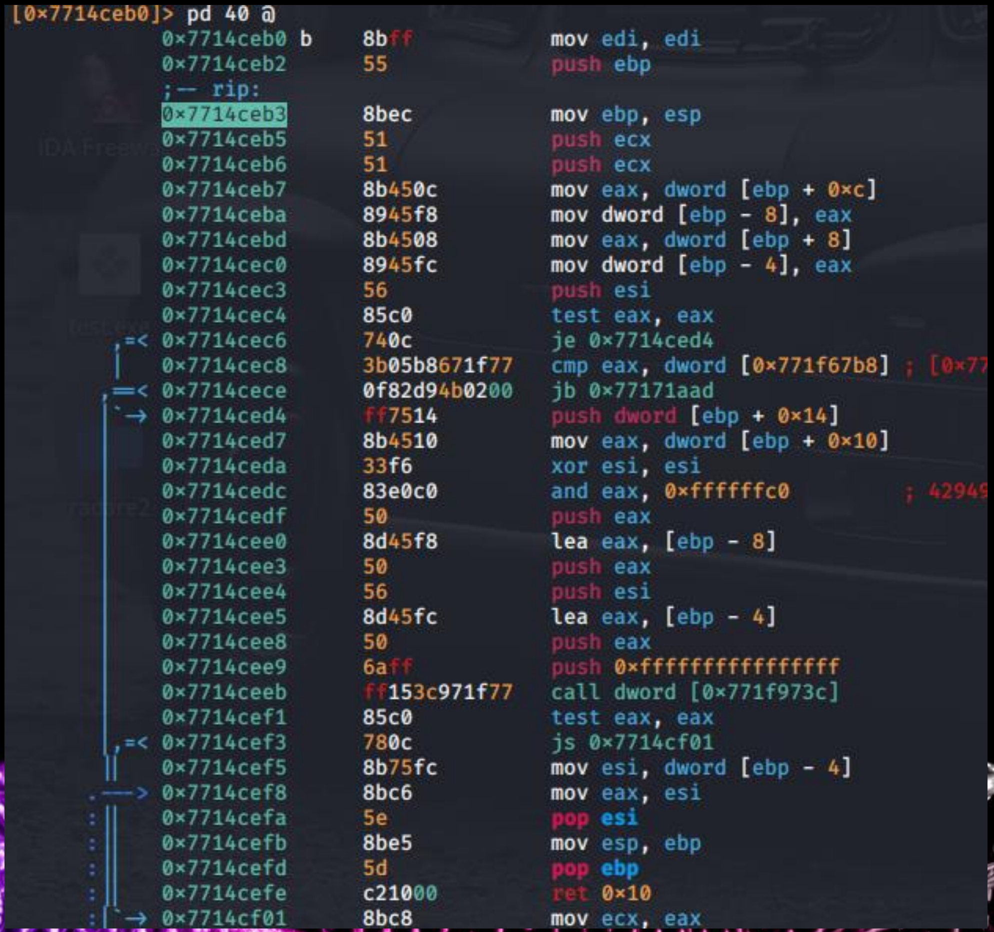
Setting breakpoints and try to hit them

```
[0x0040da51]> db 0x7714ceb0
[0x0040da51]> db 0x7714c170
[0x0040da51]> dc
(2480) loading library at 000000006B370000 (C:\Windows\SysWOW64\ntdsapi.dll) ntdsapi.dll
(2480) loading library at 0000000075470000 (C:\Windows\SysWOW64\ws2_32.dll) ws2_32.dll
hit breakpoint at: 7714ceb0
[0x7714ceb0]>
```

Execuția s-a oprit la adresa lui VirtualAlloc, deci este apelat, în continuare se caută ret și se pune breakpoint pentru investigare zona intoarsa de VirtualAlloc

(in eax)

Hit2



```
[0x7714ceb0]> pd 40 @
0x7714ceb0 b  8bff          mov edi, edi
0x7714ceb2    55            push ebp
;-- rip:
0x7714ceb3    8bec          mov ebp, esp
0x7714ceb5    51            push ecx
0x7714ceb6    51            push ecx
0x7714ceb7    8b450c        mov eax, dword [ebp + 0xc]
0x7714ceba    8945f8        mov dword [ebp - 8], eax
0x7714cebd    8b4508        mov eax, dword [ebp + 8]
0x7714cec0    8945fc        mov dword [ebp - 4], eax
0x7714cec3    56            push esi
0x7714cec4    85c0          test eax, eax
0x7714cec6    740c          je 0x7714ced4
0x7714cec8    3b05b8671f77  cmp eax, dword [0x771f67b8] ; [0x77
0x7714cece    0f82d94b0200  jb 0x77171aad
0x7714ced4    ff7514        push dword [ebp + 0x14]
0x7714ced7    8b4510        mov eax, dword [ebp + 0x10]
0x7714ceda    33f6          xor esi, esi
0x7714cedc    83e0c0        and eax, 0xffffffc0          ; 42949
0x7714cedf    50            push eax
0x7714cee0    8d45f8        lea eax, [ebp - 8]
0x7714cee3    50            push eax
0x7714cee4    56            push esi
0x7714cee5    8d45fc        lea eax, [ebp - 4]
0x7714cee8    50            push eax
0x7714cee9    6aff          push 0xffffffffffffffff
0x7714ceeb    ff153c971f77  call dword [0x771f973c]
0x7714cef1    85c0          test eax, eax
0x7714cef3    780c          js 0x7714cf01
0x7714cef5    8b75fc        mov esi, dword [ebp - 4]
0x7714cef8    8bc6          mov eax, esi
0x7714cefa    5e            pop esi
0x7714cefb    8be5          mov esp, ebp
0x7714cefd    5d            pop ebp
0x7714cefe    c21000        ret 0x10
0x7714cf01    8bc8          mov ecx, eax
```

In final se observa ret

Se pune breakpoint la adresa, se

Continua execuția si se urmaresc registrii

Cu ajutori comenzii dr

Q/A