Malware Analysis

Agenda

Malware vs Incident Response

Reporting

Lab Creation

The Malware Analysis process

Code Reversing

Framework

Windows Architecture Primer

Assembly Language Primer

Demo

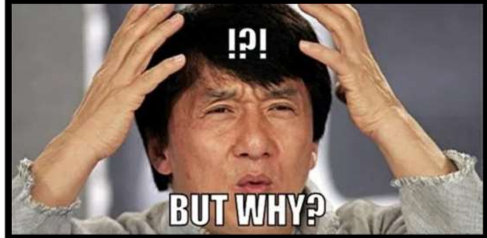
Malware vs Incident Response

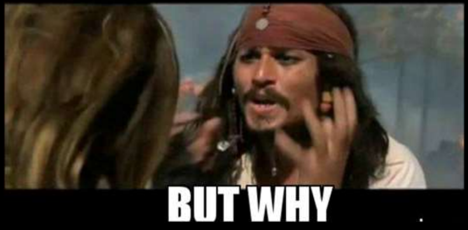
- Malicious software is an integral component of many security incidents and the associated investigations
- Organizations often struggle to understand malware they encounter during incident response
- What is "**Incident Response**" == is the methodology an organization uses to respond to and manage a cyberattack
- A lot of debate when it comes to define malware
- We will use the **NIST SP 800-83** definition:
  - *"**Malware**, also known as **malicious code**, refers to a <u>program</u> that is <u>covertly</u> inserted into another program with the <u>intent</u> to destroy data, run destructive or intrusive programs, or otherwise compromise the confidentiality, integrity, or availability of the victim's data, applications, or operating system."*

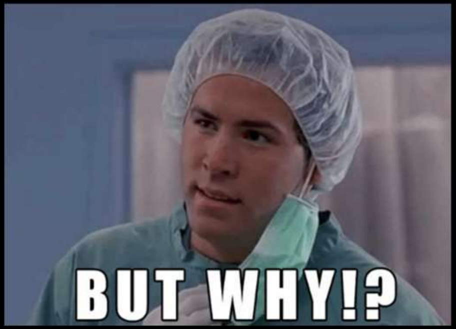
NIST: Guide to Malware Incident Prevention and Handling for Desktops and Laptops

But why?

- Learn core malware analysis techniques so you can:
  - Determine the nature of malware threats
  - Identify the scope of the incident
  - Eradicate malicious artifacts
  - Fortify system and network defenses
  - Enhance your ability to handle malware incidents
- Answer critical questions:
  - What are the malware characteristics?
  - What threat does it poses?
  - How skilled is the adversary using it?
  - How do you contain and recover from the incident?
  - What goals might he be pursuing?
  - How do you strengthen system and network defenses?







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

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

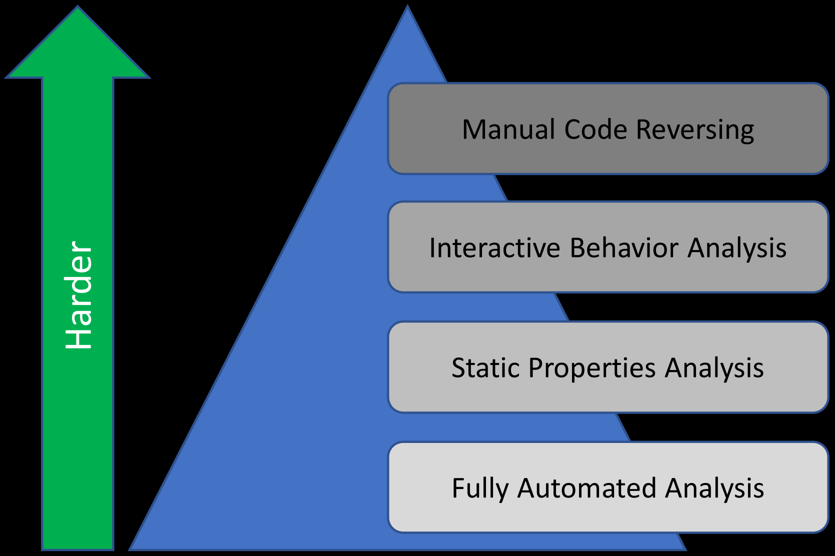
Lab Creation

- To analyze malware safely and effectively we need a 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
- Physical vs Virtualized environment
  - A physical environment simulates better but 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

The Malware Analysis Process

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

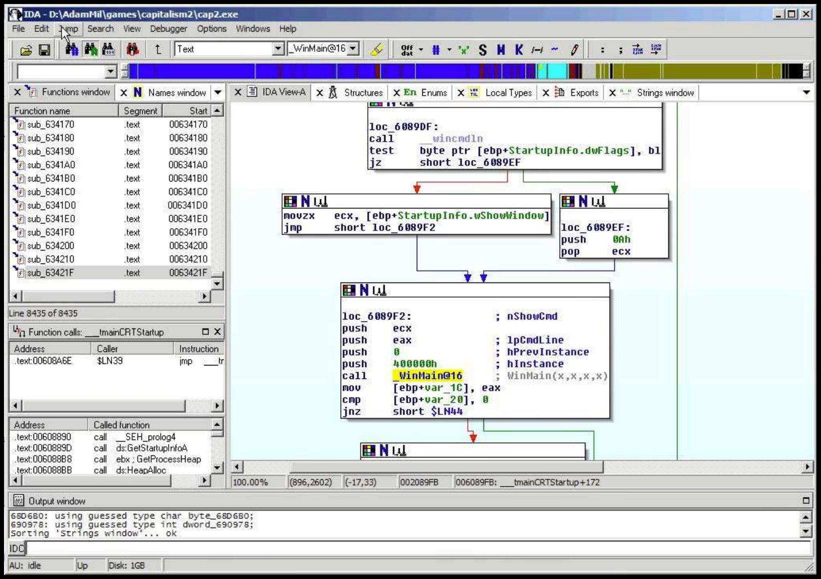
1. **Automatic Analysis**
   - The easiest way to begin investigating a specimen is to examine it using fully automated tools. These usually do not provide the same insight as a human analysis would, but contribute to the IR process by rapidly handling vast amounts of specimens
2. **Static Analysis**
   - 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 Process

3. **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**



Function name	Segment	Start
sub_634170	.text	00634170
sub_634180	.text	00634180
sub_634190	.text	00634190
sub_6341A0	.text	006341A0
sub_6341B0	.text	006341B0
sub_6341C0	.text	006341C0
sub_6341D0	.text	006341D0
sub_6341E0	.text	006341E0
sub_6341F0	.text	006341F0
sub_634200	.text	00634200
sub_634210	.text	00634210
sub_63421F	.text	0063421F

Line 8435 of 8435

Function calls: ___tmainCRTStartup

Address	Caller	Instruction
.text:00608A6E	$LN39	jmp ___tr

Address	Called function
.text:00608890	call __SEH_prolog4
.text:0060889D	call ds:GetStartupInfoA
.text:006088B8	call ebx ; GetProcessHeap
.text:006088BB	call ds:HeapAlloc

```
loc_6089DF:
call    __wincmdln
test    byte ptr [ebp+StartupInfo.dwFlags], bl
jz      short loc_6089EF
```

```
movzx   ecx, [ebp+StartupInfo.wShowWindow]
jmp     short loc_6089F2
```

```
loc_6089EF:
push    0Ah
pop     ecx
```

```
loc_6089F2:                 ; nShowCmd
push    ecx
push    eax                 ; lpCmdLine
push    0                   ; hPrevInstance
push    400000h             ; hInstance
call    _WinMain@16         ; WinMain(x,x,x,x)
mov     [ebp+var_1C], eax
cmp     [ebp+var_20], 0
jnz     short $LN44
```

```
100.00%  (896,2602)  (-17,33)  002089FB  006089FB: ___tmainCRTStartup+172
```

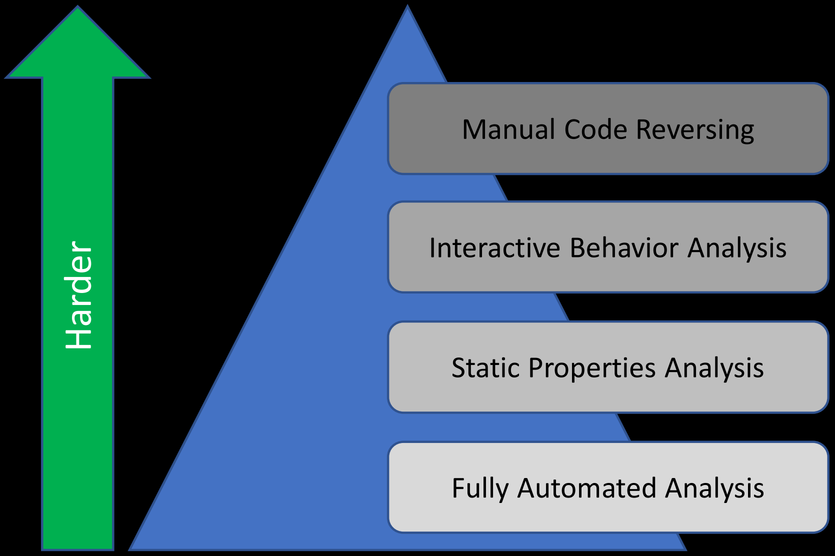
```
68D6B0: using guessed type char byte_68D6B0;
690978: using guessed type int dword_690978;
Sorting 'Strings window'... ok
```

```
IDC
AU: idle   Up   Disk: 1GB
```

The Malware Analysis Process

5. **Interactive Behavioral Analysis**
   - The final step in the analysis process is to interact inside the isolated environment with the specimen by simulating, inside the laboratory, an environment connected to the "Internet".
   - The objective is to enact a response from the specimen in order to fully understand the capabilities and its objectives.

- Each phase adds context and information for the following one and helps the analyst to be more efficient
- There are situations where phases might be skipped, if enough date is present, but is strongly recommended to follow the process as presented



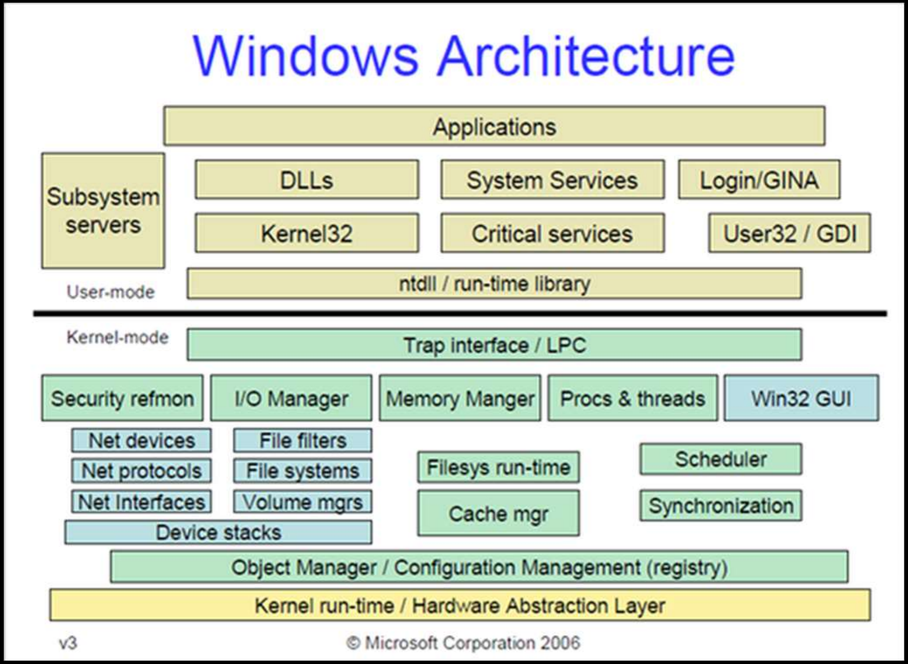
Code Reversing

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.**

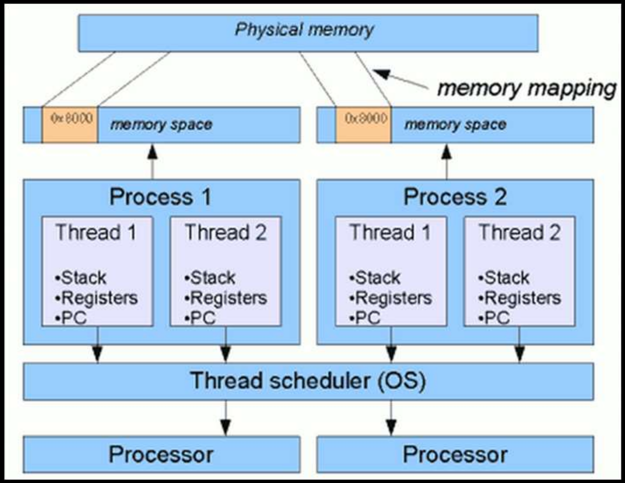
Windows Architecture Primer

- 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** - a page table 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)?**



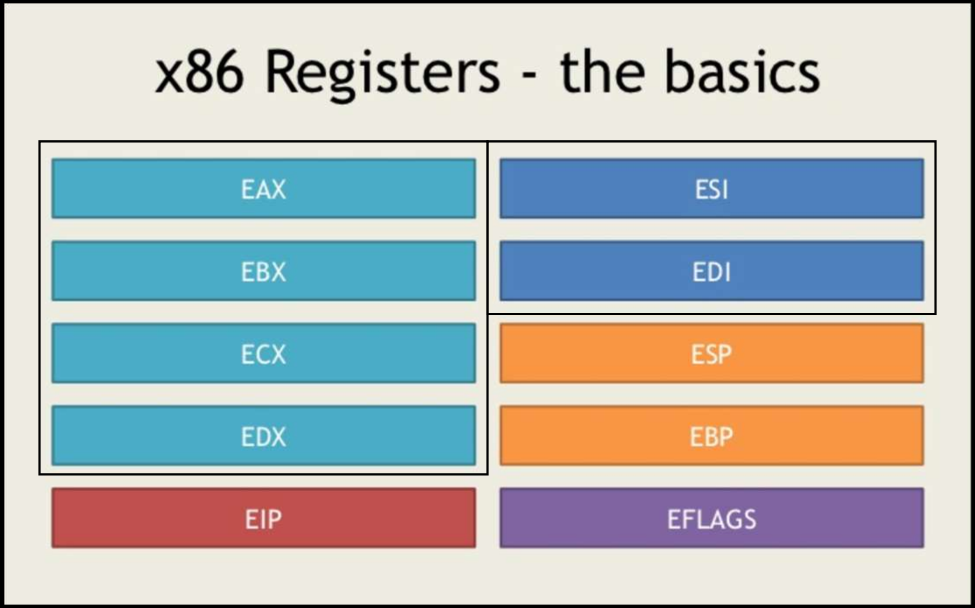
Windows Architecture Primer

- **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.



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

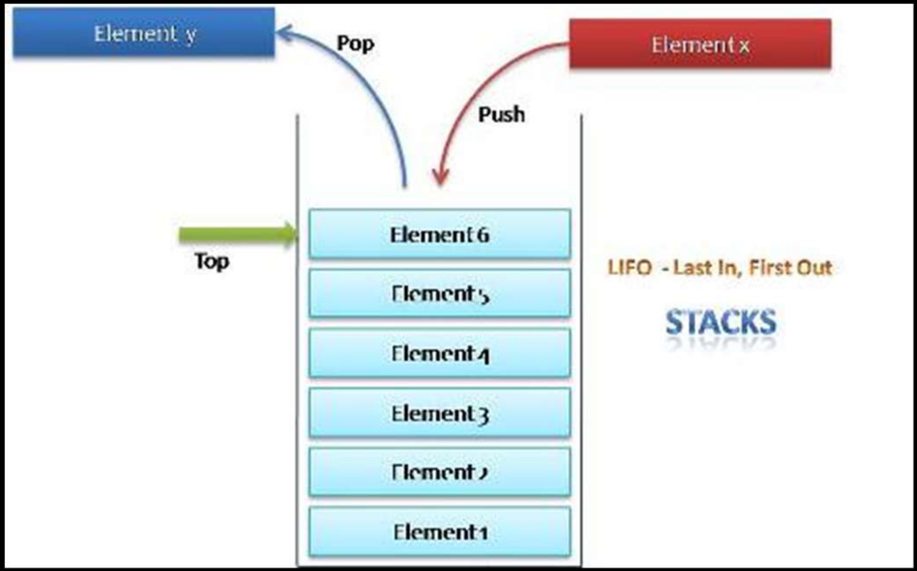
- Segment Registers:
  - Stack Segment (SS). Pointer to the stack.
  - Code Segment (CS). Pointer to the code.
  - Data Segment (DS). Pointer to the data.
  - Extra Segment (ES). Pointer to extra data ('E' stands for 'Extra').
  - F Segment (FS). Pointer to more extra data ('F' comes after 'E').
  -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



Demo Time

- Malware Analysis
- Anti-Analysis Techniques: Reverse Engineering
- Homework?

- Home Work Time

Thank you