Malware Analysis

- process of understanding how a suspected malware behaves, & its goal.
- analyzed samples may be in the form of a file, a URL, or the memory of a compromised system.
- Use cases :-
  1. Malware Detection
  2. Incidence Response Engagements.
  3. Threat Hunting
- Types of Malware Analysis :-

Static Analysis	Dynamic Analysis
1. Analysis without running.	1. Analysis by running code in a controlled environment
2. Speculate behavior from text strings found in the file.	2. Observe behavior while running the file.
3. Use a disassembler or a decompiler to reverse-engineer a binary.	3. Use disassembler, de-compiler & debugger to rev-eng a binary.
4. Code & data not easily seen if obfuscated.	4.

- Stages in conducting Malware Analysis :-

① Fully Automated Analysis → ② Static Properties Analysis → ③ Interactive Behavior Analysis → ④ Manual Code Reversing

① Fully Automated - run sample in a sandbox & analyze logs.

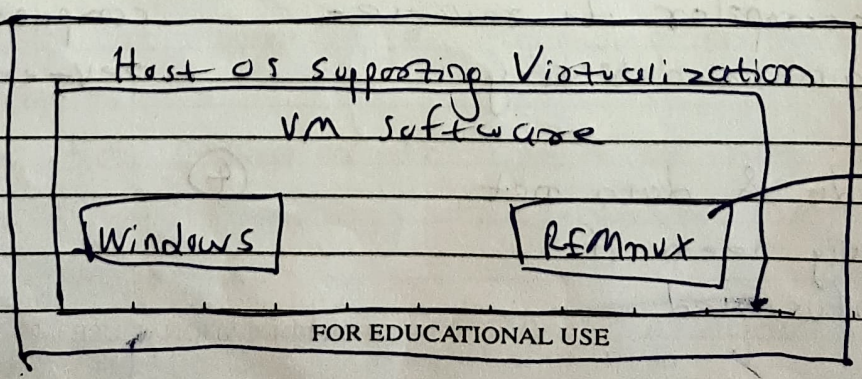
② Static Properties - identify IOCs from file properties and strings.

③ Interactive Behavior - run sample in a controlled environment while monitoring system changes at runtime.

④ Manual Code Reversing - debug the malware and understand capabilities that might not be shown during automated & interactive analysis.

- Lab setup :-

- Prerequisites - virtualization feature should be enabled in the system's BIOS



- Running the Flare-VM script to install the tools that we need on windows. MS office recommended, but not compulsory.

- VirtualBox: -
  - install guest addition tools.
  - N/w setup - NAT

- Disable updates from apps & O.S, turn off security monitoring tools.

- Install Windows 10 image from developer.microsoft.com
- Download VirtualBox OVA for Remnux from remnux.org

- Increase video memory for Win10.

Flare VM script execution steps:-

1. Go to mandiant/flare-vm (github) & download the code.
2. Open powershell window as admin.
3. Enable powershell scripts to execute - execute the foll. command:-
   set-executionpolicy unrestricted.
4. Go to the folder where flare VM is installed & install it with the following command:-
   .\install.ps1
5. Select the Attached to optim as not attached under network settings to disable network communication.
6. Don't forget to take snapshots of your state at frequent intervals.

- Flare VM :-
  - windows-based distribution used for malware analysis.
  - Installer is a powershell script.
  - Uses Chocolatey to deploy tools.
    - open src windows package manager, like apt-get for Linux & Brew for MacOS.
    - Makes it easy to install, update, and manage software

- Tools Used in this Course :-

Static Analysis :-	Dynamic Analysis :-	Manual Code Reversing :-
• Hash Calc	• RegShot	• x64 dbg
• Detect-it-Easy	• Procmon	• WinDbg
• Strings	• Wireshark, Fiddler	• OllyDebug
• PE-Bear, PEStudio	• Autoruns	
• CyberChef		

- Scope of this course :-

Tools Used	Files types discussed :-
• Detect-It-Easy	• Identify Hash (md5, sha256)
• PE-Bear	• Type of file (.exe, .dll, office document, script, etc)
• CyberChef	• File details : Time Stamp, compiler
• HashCalc	• Text strings (URLs, filenames, registry)
• HxD	• Imported API call
• Flare-VM	
• Fakenet-NG	

- Static Analysis
  - Gather information about the file without running the sample.
  - Hash Tools - Identify hash (MD5, SHA256)
  - Type of file - exe, dll, etc
  - Obfuscated? Packed?
  - File Details - time stamp, compiler, company name, digital signatures
  - Text strings - URLs, filename, registry
  - PE file details - Imported API calls
  - Hashing Tools → HashCalc, Md5sum, sha256sum
    for linux (command-line tools)
  - Tools that identify file types → DiE (Detect it Easy), PEiD

Identifying the file type tells us what tools we will use for further analysis. ex: if a file was identified as a word doc, we would use a tool that can identify if the document contains a macro code or not.

  - Tools to id strings - Strings

strings are used to identify things such as URLs, IPs,

- To see the activities that happened during execution of the sample, we have to use behavioral monitoring tools.

② ProcMon (Process Monitor)
- Behavioral Monitoring Tool that logs all processes, file and registry changes.
- Filtering feature - helps us to see changes made only by the target sample.
- We can see which API functions were used, the text arguments passed to those functions.
- Resulting log can be used to identify the sequence of APIs used and determining the behavior of the target sample.

③ Wireshark :-
- Network traffic monitoring tool.
- logs network traffic in the form of packets
- Huge list of supported protocols such as TCP, UDP, HTTP, etc.

④ Fiddler :-
- Network traffic monitoring tool focused on web-based traffic like HTTP.
- Shows different views of the HTTP header and data.
- MITM feature - used to inspect data from an HTTPs traffic.

(5) Fakenet :-
- Network traffic monitoring tool.
- Most useful when we want to monitor outgoing traffic from the sample we are analyzing.

(6) Autoruns :-
- Used to identify where the malware installed itself.

* Demo
(1) re_whatami.exe
Part1 :-
Tools used → Regshot, Procmon & Wireshark.

* Manual Code Reversing :-

- Static & Dynamic Analysis gives a lot of information about how a malware behaves. But there is a possibility of other behaviors that may only trigger on certain conditions. Hence, the need for manual code reversing.

- PE file → most common file analyzed by a malware reverse engineering.

→ Debugger :-

- Tools → x86dbg, IDA Pro (commercial)

- Requirements :- Assembly language, Environment APIs, C language.

→ Decompiler :-

- used to convert binary files into some form of code that can be read by humans.

- Tools → Hex-Rays, dnSpy (.NET decompiler)

→ Demo 1

⇒ Analyzing Powershell scripts

- Method 1: Static Analysis ⇒ Use Text Editor
- Powershell ISE: Debug } Method 2.
(pre installed in Windows)
- Powershell scripts are used mainly in the post-exploitation phase.
- Common functions & methods ⇒ set alias
IEX (invoke expression)

⇒ Analyzing Javascript ~~code~~ files:-

- look for a to b functions.
- look for eval functions.
- " " document.write().
- Javascript ~~code~~ executes immediately. To prevent this, use visual studio code to debug the .js file.

⇒ Analyzing Macro-enabled documents.

- Tools ⇒ Oletools, MS Word (Excel Macro Editor)

⇒ Linux ELF File Analysis.

- using Remnux.
- Tools:-

(1) Linux Native: strace (System trace), ltrace (local trace), gdb (GNU debugger)

(2) IDA.

→ Analysing AspX Webshell

- When a attacker compromises a web servers, the most likely malware that gets deployed is a webshell.

- Webshell - Interface that gives an attacker remote access to web server.
The webshell src code is located at the server. When visited from an internet browser, you will only see the HTML code of the source, but not the scripts behind the webshell.

- Features of a webshell: Shell commands, upload files, download files, execute files, list files & folders, delete files

- AspX files are commonly used in an IIS web server..

- Analysing .aspx files :- (to not attached)
(1) turn off network adaptor in VM's settings
(2) Go to control Panel → Turn Windows Features on or off → Internet Information services → WWW services → Application Development Features → select the most recent version of ASP.NET → hit okay button.
(3) open IIS manager
(4) right click on the server in the left panel & start the servers if not already started

(5) Expand the server until you reach Default Website.
(6) Right click on Default Website & select Manage Website & click on Explore.
(7) Then, right click on Default Website & select Explore. (A folder will open where all the web server files are located.
(8) Copy aspx file to be analyzed to this folder.
(9) Open browser & in the address bar, type localhost/file-name.aspx.
This will open the webshell.

Instead of going through this process of setting up IIS server, we can use a tool called Abyss Web Server.

- Configuring Abyss Web Server :-
(1) Right click on the icon in the system tray & select restore.
(2) Select Hosts → Default Host → Configure
A webpage will open.

→ Analyzing JAR Files :-
- Java Archives or JAR is a cross-platform file format that can be used on Windows, Linux, Mac, Android, and many more.

- Because of this cross-platform benefit threats actors can compromise multiple endpoints of different operating systems.

- A JAR file is basically a zip archive.
- Tools - bytecode viewer, jd gui, eclipse, net pins, visual studio code with java extension.

⇒ Recap:-
- Malware Analysis & the process
- How to set up a malware analysis environment. (windows)
- Conducting static, dynamic, and manual code analysis.
- Analysis of various file types:-
  - Powershell scripts
  - Javascripts
  - Java Archives
  - linux native executable (ELF file)
  - word documents

Notes while analysing malware:-
- Properly configure VM resources to prevent compromising the VM host and network.
  e.g. set network to host-only, set read-only shares.
- Update hypervisor
- Avoid accidental malware execution - rename malware filename extensions or password protect samples.
- Take frequent snapshots.

* Assembly Language :- (x86 Assembly)
→ aka "asm" ; low-level programming language.
→ Usages :-
- hardware manipulation
- specialized processor instructions.
- address critical performance issues.

- Actual raw instructions that the C.P.U executes are called machine codes.
Assembly is the human-readable counterpart of machine codes/language.

- needs an assembler.

- Examples that use Assembly programming :-
- Nintendo Entertainment System
- Robot Tron 2084 (an old arcade game)
- MS-DOS & IBM DOS O.S's.
- different device drivers
- computer viruses.

- Why learn Assembly ?
- for speed or memory optimizations.
- for debugging high-lvl programming languages
- for Reverse Engineering.

→ X86 Memory Architecture: -

- Little-endian - values are written LSByte first
ex: The 32-bits double word 0x1BA583D4 (the 0x denotes hexadecimal) would be written in memory as: [D4] [83] [A5] [1B]

- An X86 instruction can have 0 to 3 operands
- Operands are separated by commas & usually follow the format: -

mnemonic destination, source

ex: mov eax, 0FE00A756
mov → mnemonic
eax → Dest. Op #1
0FE00A756 → source operand #2.

- Opcode & mnemonics: -
- Opcode = operation code; the number interpreted by the hardware that represents the operation to perform.
- mnemonic = reserved name for family of opcodes.

→ X86 Registers: -

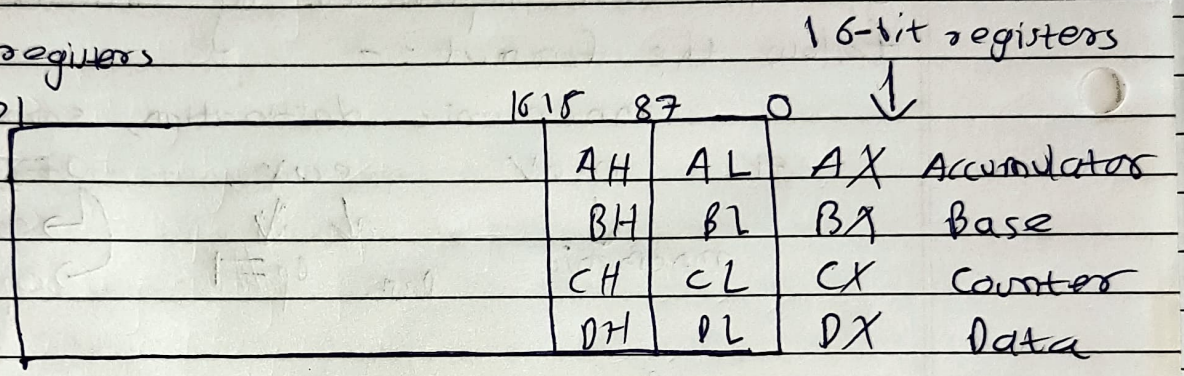
- To speed up the processor operations, the processor includes some internal memory storage locations called registers.
- These registers store data elements for processing to eliminate the need to access memory.

① Data Registers :-

- used for arithmetic, logical & other operations
- EAX, EBX, ECX, and EDX.

AX - Primary Accumulator.
BX - Base Register.
CX - Count Register.
DX - Data Register.

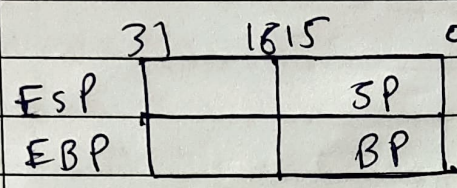
- 32-bit registers



- Lower half of the 32-bit registers can be used as 4 16-bit data registers - AX, BX, CX, DX. The lower & higher halves of the above mentioned 4 16-bit registers can be used as 8-bit data registers such as AH, AL, BH, BL, CH, CL, DH & DL, where H means high & L means low.

AX → used in I/O & most arithmetic instructions.
DX → used for index addressing
CX → used to store loop count on iterative operations.
DX → also used in I/O operations.

② Pointer Registers
- EIP, ESP and EBP
IP - Instruction Pointer
SP - Stack Pointer
BP - Base Pointer



- EIP → Instruction Pointer ⇒ stores the offset address of the next instruction to be executed.
ESP → Stack Pointer ⇒ provides the offset value within the program stack, which always points to the top of the stack
EBP → Base Pointer ⇒ helps in referencing the parameter variables passed to a subroutine which always points to the base of the stack.

③ Index Registers: -
- ESI & EDI
- Non-volatile general purpose registers, often used as a pointer.
- ESI → used as a source index for string ops.
EDI → " " dest. index " "

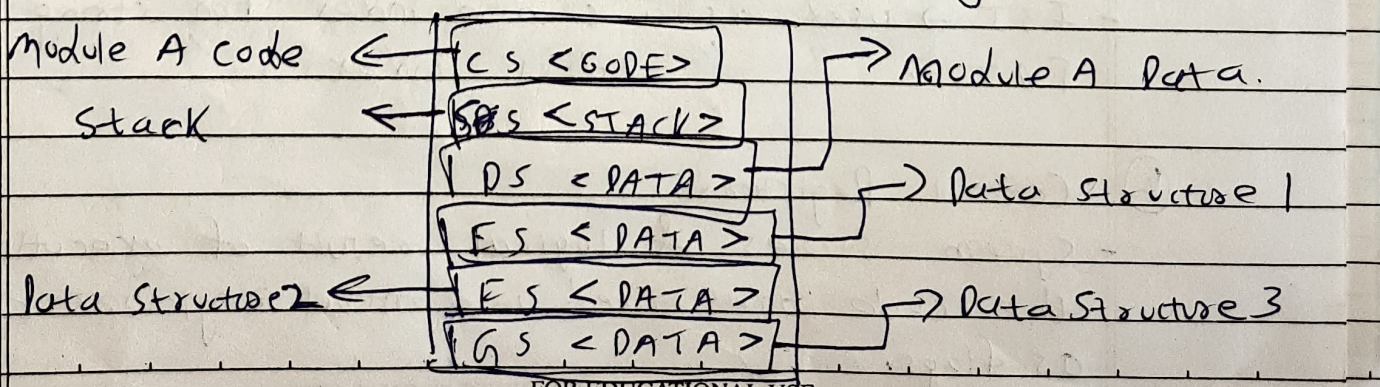
④ Control Registers.
- System states & logical result of executing the code are stored in control registers known as flags.

- Every bit of the flag register has its own purpose.

Flag :					O	D	I	T	S	Z		A		P		C	
Bit no. :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

OF - Overflow flag
DF - Direction Flag
IF - Interrupt Flag
TF - Trap Flag
SF - Sign Flag
ZF - Zero Flag
AF - Auxiliary Carry Flag
PF - Parity Flag
CF - Carry Flag

(5) Segment Registers :- (segments are specific areas defined in a program for containing code, data, and stack.
- CS, DS, SS, ES, FS & GS.
CS - code segment
DS - Data segment
SS - Stack segment
ES (Extra segment), FS and
GS - additional segments for storing data.



→ Stack :-

- used for temporary storage of data.
- range of memory.
- usually used to pass arguments to functions & procedures within the program.
- can also be used to temporarily save register values.
- makes use of 2 instructions :-

① push - to store value in stack.
② Pop - to remove a value from stack.

→ Common Assembly language Instructions :-

1) mov.

- used to move data.
- mov destination, source.
- Immediate :-

```
mov eax, 0xabcd1234
```

Register - to - Register :-

```
mov eax, ebx
```

Memory :-

```
mov eax, dword ptr[ebp - 0xc]
```

EAX = 0xDEADBEEF

2) ADD / SUB

- used to perform simple addition / subtraction.

ADD / SUB destination, source.

- Example:-

```
mov eax, 0x00000010
mov ebx, 0x00000020
add eax, ebx    // EAX= 0x00000030
sub ebx, 0x0F   // EBX = 0x00000011
```

3) INC / DEC

- used for incrementing / decrementing an operand by one.

INC / DEC destination.

- Example:

```
mov eax, 0x00000010
mov ebx, 0x00000020
inc eax          // EAX = 0x00000011
dec ebx          // EBX = 0x0000001F
```

4) XOR

- implements bitwise XOR operation

XOR destination, source

- Example:-

```
mov eax, 0x00000010    // 0010 0000
mov ebx, 0x00000018    // 0001 1000
xor eax, ebx
```

5) NOP
- used for timing purposes or to force memory alignment.
- nop.

6) Unconditional Jump
- JMP instruction provides an unconditional transfer of control - may be forward or backward.
- JMP label.
- Example:-

```
jmp   L20
mov   eax, 0x00000010
L20:
mov   eax, 0xF0000000
EAX = 0xF0000000
```

7) Conditional Jump
- The control flow is transferred to a target instruction if a specified condition is satisfied.

j <condition> label

Instruction	Description	Flag tested
JE/JZ	Jump Equal or	ZF = 1
JNE/JNZ	Jump zero	ZF = 0

8] cmp

- generally used for conditional execution, compares two operands.

- cmp destination, source

- Example:-

```
mov   eax, 0
inc   eax
cmp   eax,1        //sets ZF to 1
jnz   LTRUE        //does not
jump
dec   eax
ltrue:
nop                //EAX=0
```

9] Loop:-

- transfers the control flow to the indicated label & decrements ECX register by 1 until the counter register reaches the value 0.

- loop label

- Example:-

```
mov   ecx, 0x10
L2:
<loop body>
loop  L2
```