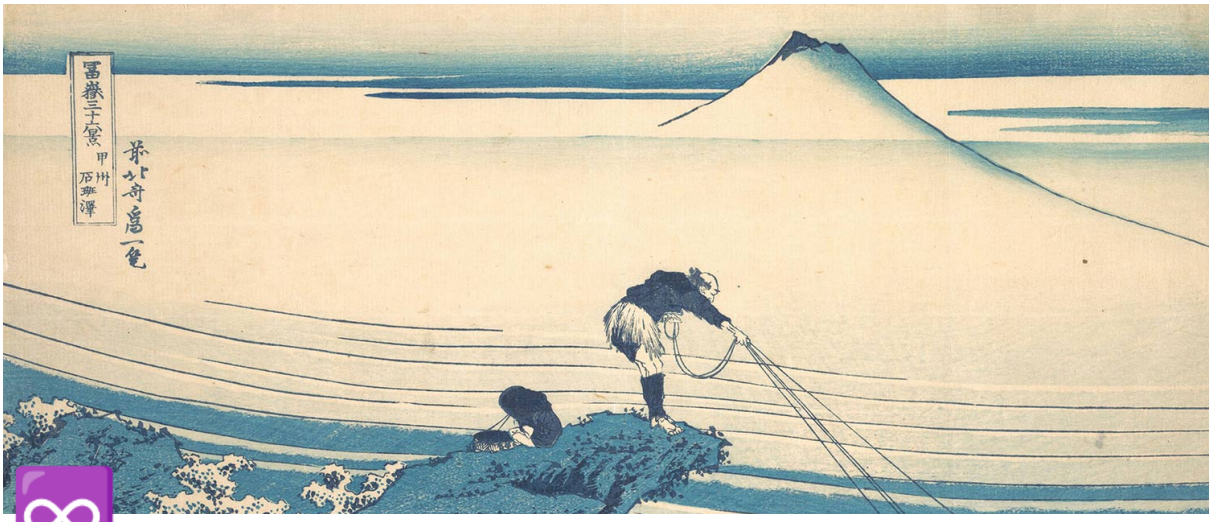

Reverse Engineering & Malware Analysis

- **Malware:**
  - Malware refers to malicious executable binaries used by attackers to perform a range of malicious actions. These actions can include spying through keyloggers or Remote Access Trojans (RATs), data deletion, and data encryption for ransom.

- **Types of Malware:**
  - Malware comes in various types, each with specific functionalities. Common types include:
    - Trojans: Can destroy or exfiltrate data and be used for spying.
    - RATs (Remote Access Trojans): Enable remote access and command execution on systems.
    - Ransomware: Encrypts files and holds them hostage for ransom.
    - Droppers: Download and drop additional malware onto the system.

- **Malware Analysis:**
  - Malware analysis involves studying and understanding the functionality, origin, and potential impact of a given malware sample. It aims to extract valuable information that sheds light on how the malware works, how systems get infected, and how to defend against similar attacks in the future.

- **Objectives of Malware Analysis:**
  - Key objectives include:
    - Understanding the type and functionality of malware.
    - Determining the infection method (e.g., targeted attack or phishing).
    - Analyzing how malware communicates with attackers.
    - Preparing for future malware detection and signature generation.

- **Types of Malware Analysis:**
  - Various methods include:
    - Static analysis: Analyzing malware without execution to extract metadata (e.g., P.E. headers, strings).
    - Dynamic analysis: Executing malware and observing its behavior using a debugger.
    - Code analysis: Analyzing/reverse engineering the assembly code, combining static and dynamic analysis.
    - Behavioral analysis: Monitoring malware post-execution, tracking processes, registry entries, and network activities.

- **Common Steps in Malware Analysis:**
  - The process typically involves:
    - Identification: Confirming the presence of malware and understanding its characteristics.
    - Acquisition: Obtaining a malware sample while ensuring proper containment.
    - Preliminary Analysis: Gathering initial data about the malware.
    - Static Analysis: Examining the malware without execution to understand its structure.
    - Dynamic Analysis: Executing the malware in a controlled environment to observe its behavior.
    - Code Analysis: Analyzing the malware's code to uncover its functionality and logic.
    - Behavioral Analysis: Monitoring the malware's actions during execution to understand its interactions.

- Reverse Engineering: Deconstructing the malware to reveal its inner workings.
- Post-Analysis: Documenting findings, generating reports, and gaining insights for future prevention and detection efforts.

Advantages of Malware Analysis:

1. **Threat Detection:** Malware analysis helps detect previously unknown threats, enabling proactive defense against attacks.
2. **Improved Security:** By understanding malware behavior, organizations can enhance their security measures and reduce the risk of infection.
3. **Understanding Attack Techniques:** Malware analysis provides insights into attacker methods and techniques, aiding better preparation and defense against future attacks.
4. **Early Detection:** Analyzing malware in its early stages allows for mitigation and quicker recovery from attacks.
5. **Forensics:** Malware analysis can provide valuable information for forensic investigations and support the prosecution of attackers.

Disadvantages of Malware Analysis:

1. **Time-Consuming:** Malware analysis can be time-consuming and requires specialized knowledge and tools.
2. **Risk of Infection:** Analyzing malware in an uncontrolled environment may lead to its spread, potentially causing harm to other systems.
3. **Cost:** Malware analysis requires specialized tools and expertise, which can be expensive for organizations to acquire and maintain.
4. **Difficulty:** Malware is continually evolving, and the analysis process can be challenging, requiring specialized knowledge and expertise.
5. **False Positives:** Malware analysis can sometimes yield false positives, leading to false alarms and undermining confidence in existing security measures.

Static Analysis:

- Static analysis involves examining malware without executing it.
- This is done by analyzing the binary code to understand functionality and identify malicious activity.

- Static analysis is used to identify potential security threats without the risk of infecting the analysis environment.

Dynamic Analysis:

- Dynamic analysis entails executing the malware in a controlled environment and observing its behavior interactively.
- It involves monitoring file system, registry, and network activities to understand the malware's behavior in real time.

Behavior Analysis:

- Behavioral analysis observes and interacts with malware running in a lab environment.
- Analysts aim to understand registry, file system, process, and network activities to determine the malware's capabilities and objectives.

Debugging:

- Debugging is the process of identifying, isolating, and correcting problems in computer programs or systems. It involves reproducing an issue, determining its source, and implementing fixes or workarounds.

Basics of Debugging using OllyDbg:

- OllyDbg is a popular Windows debugger for malware analysis.
- Debugging aims to identify and correct issues in software or hardware.
- Debugging can be a vital part of the software development lifecycle.
- OllyDbg is a user-mode debugger used in reverse engineering.
- To learn reverse engineering, one can start with debugging using OllyDbg.
- Debugging can be used to analyze and understand the behavior of target binaries.
- Debugging can help understand and bypass security mechanisms in software.

The basics of debugging and using OllyDbg are introduced as part of malware analysis and reverse engineering. The process involves understanding how to use OllyDbg to analyze a crackme challenge binary, where various debugging concepts are applied.

Obfuscation:

- **Obfuscation** makes things harder to understand, used in programming to protect software from reverse engineering.
- An **obfuscator** is a tool that converts simple code into a complex, less readable version while preserving functionality.
- **Malware obfuscation** conceals code or data to make it challenging to interpret, used by malware authors.
- It hides critical strings in programs that might reveal malware behavior, such as registry keys or infected URLs.
- **Packed programs** are a type of obfuscated program, making it resistant to analysis, especially static analysis.
- Few strings in the code can indicate malware obfuscation, as non-malicious programs often have more strings.

Malware Obfuscation Techniques:
Malware writers frequently employ obfuscation techniques to evade antivirus scanners and security measures. Some common obfuscation techniques used in malware include:

1. **Dead-Code Insertion:** Inserting non-functional or "dead" code (e.g., NOP instructions) to change the appearance of the program while retaining its functionality. This can confuse signature-based antivirus scanners.
2. **XOR:** Concealing data by swapping the contents of two variables in the code using exclusive OR (XOR) operations.
3. **Register Reassignment:** Switching registers between instructions while keeping the program's functionality intact.
4. **Subroutine Reordering:** Rearranging the order of subroutines or functions in the code to disrupt the program's flow.
5. **Instruction Substitution:** Replacing specific instructions with equivalent ones, making analysis more challenging.
6. **Code Transposition:** Changing the order of code instructions or blocks to obfuscate the program's structure.
7. **Code Integration:** Combining multiple code fragments to make the code harder to follow.
8. **Base64 Encoding:** Encoding data, such as strings, in Base64 format, which can hide their actual content.

9. **Packers:** Using packing techniques to compress and protect the malware, making it resistant to analysis.

These techniques make it more difficult for security analysts and antivirus tools to detect, analyze, and understand the behavior of malicious software.

Packers:

- Packers are used to compress the content of the malware file.
- Packers can make malware appear different from its true form.
- Common tools for packing and unpacking malware include UPX and EXEinfo.

Packing Process:

1. The original code is uploaded into the packer tool and goes through the packing process to compress or encrypt the data.
2. The original PE header and code are compressed or encrypted and stored in the packed section of the new executable.
3. The packed file consists of a new PE header, packed section(s), and a decompression stub used to unpack the code.
4. During the packing process, the original entry point is relocated or obfuscated in the packed section, making code analysis challenging.
5. The decompression stub is used to unpack the code upon delivery.

Common Object File Format (COFF):

- COFF is a format for executable and shared library files used on Unix systems.
- The ELF format is used for Linux.
- The PE file format falls under COFF.
- The PE data structures include DOS Header, DOS Stub, PE File Header, Image Optional Header, Section Table, Data Dictionaries, and Sections.

DOS Header:

- The DOS Header occupies the first 64 bytes of the file and begins with the ASCII strings "MZ."

- "MZ" represents "Mark Zbikowski," one of the developers of MS-DOS.
- The "e_magic" field contains the vital "magic number" (0x5A4D) to identify an MS-DOS-compatible file type.
- The "e_lfanew" field is a 4-byte offset that indicates the location of the PE Header.

DOS Stub Program:

- A stub is a small program that runs when an application starts.
- The DOS stub prints the message "This program cannot be run in DOS mode" for incompatible programs.
- It is executed by MS-DOS when a Windows-based program is loaded.

PE File Header:

- The PE header location is determined by the "e_lfanew" field in the MS-DOS Header.
- The PE header contains SIGNATURE, IMAGE_FILE_HEADER, and IMAGE_OPTIONAL_HEADER.
- The SIGNATURE is a 4-byte DWORD signature.

DLL (Dynamic Link Library)

- **Definition:** A file format for holding code and data that can be used by multiple programs simultaneously.
- **Reusability:** DLLs contain functions and procedures for reuse by different applications, reducing code duplication.
- **Dynamic Linking:** Code in DLLs is loaded into memory when an application runs, as opposed to static linking at compile time.
- **Modularization:** DLLs allow software to be divided into manageable, independent components.
- **Shared Resources:** DLLs can store and share resources such as icons, images, and configuration data.
- **Version Control:** Developers can release updated DLLs for bug fixes or new features without recompiling entire applications.
- **Load-time and Run-time Linking:** DLLs can be linked at application load time or dynamically while the application runs.

- **APIs:** Windows functions and libraries are often implemented as DLLs, allowing developers to use these functions.
- **Security:** Malicious DLLs can pose security risks if loaded by an application.
- **File Extension:** In Windows, DLLs typically have a ".dll" file extension.

DLLs are essential for creating modular, efficient, and maintainable software on the Windows operating system and beyond.

Dynamic Malware Analysis Tools:

- **ProcMon:** A free tool developed by Windows SysInternals for monitoring Windows filesystem, registry, and process activity in real-time. It combines 2 legacy tools: FileMon and RegMon. Procmon offers non-destructive filtering of data and boottime logging.
- **Process Explorer:** A free tool from Microsoft used when performing Dynamic Malware Analysis. It's used to monitor running processes and shows you which handles and DLLs are running and loaded for each process.
- **RegShot:** An open-source utility to monitor your registry for changes by taking a snapshot, which can be compared to the current state of your registry. It allows you to see the changes made to your registry after the malware has been executed on your system.
- **Wireshark:** Wireshark can be used for live packet capturing, deep inspection of hundreds of protocols, browsing and filtering packets. It's multiplatform. When performing Dynamic Malware Analysis, Wireshark can be used to inspect packets and log network traffic to files.

Network Sniffing:

- A sniffer normally turns the NIC of the system to the promiscuous mode so that it listens to all the data transmitted on its segment.
- The promiscuous mode refers to the unique way of Ethernet hardware that allows an NIC to receive all traffic on the network, even if it is not addressed to this NIC.
- By default, an NIC ignores all traffic that is not addressed to it, which is done by comparing the destination address of the Ethernet packet with the MAC address of the device.
- A sniffer can continuously monitor all the traffic to a computer through the NIC by decoding the information encapsulated in the data packets.

PASSIVE SNIFFING:

- In passive sniffing, the traffic is locked but it is not altered in any way.
- Passive sniffing allows listening only.
- It works with the Hub devices.
- On a hub device, the traffic is sent to all the ports.
- In a network that uses hubs to connect systems, all hosts on the network can see the traffic.
- Thus, an attacker can easily capture traffic going through.

Most modern networks use switches. Hence, passive sniffing is no more effective.

ACTIVE SNIFFING:

- In active sniffing, the traffic is not only locked and monitored, but it may also be altered in some way as determined by the attack.
- Active sniffing is used to sniff a switch-based network.
- It involves injecting Address Resolution Packets (ARP) into a target network to flood the switch Content Addressable Memory (CAM) table.
- CAM keeps track of which host is connected to which port.

Techniques:

- MAC Flooding
- DHCP Attacks
- DNS Poisoning
- Spoofing Attacks
- ARP Poisoning

Malware authors create programs at the high-level language level and use a compiler to generate machine code to be run by the CPU.

Conversely, malware analysts and reverse engineers operate at the low-level language level; we use a disassembler to generate assembly code that we can read and analyze

to figure out how a program operates.

---

x86 Architecture:

3 components - CPU, RAM, I/O

- CPU Executes Code
- RAM stores all data and code
- I/O interacts with hardware
- The control unit gets instructions to execute from RAM using a register (the instruction pointer), which stores the address of the instruction to execute.
- Registers are the CPU's basic data storage units and are often used to save time so that the CPU doesn't need to access RAM.
- The arithmetic logic unit (ALU) executes an instruction fetched from RAM and places the results in registers or memory. The process of fetching and executing instruction after instruction is repeated as a program runs.

REGISTERS:

A register is a small amount of data storage available to the CPU, whose contents can be accessed more quickly than storage available elsewhere.

x86 processors have a collection of registers available for use as temporary storage or workspace.

- General registers are used by the CPU during execution. i.e EAX (Extended Accumulator)
- Segment registers are used to track sections of memory. i.e CS (Code Segment) and DS (Data Segment)
- Status flags are used to make decisions. i.e ZF (Zero Flag) and CF (Carry Flag)
- Instruction pointers are used to keep track of the next instruction to execute. i.e EIP (Extended Instruction Pointer) and PC (Program Counter)
- ZF (Zero Flag): Set when the result of an operation is zero, otherwise cleared.
- CF (Carry Flag): Set when the result is too large or too small for the destination operand, otherwise cleared. It indicates overflow or underflow.

- SF (Sign Flag): Set when the result is negative or when the most significant bit is set after an arithmetic operation.
- TF (Trap Flag): Used for debugging, causing the CPU to execute one instruction at a time when set.

Stack:

- Data structure with LIFO (Last In, First Out) behavior.
- Used for short-term storage.
- Stores local variables, parameters, and return addresses.
- Crucial for data management between function calls.

Function Calls:

- Functions are code segments for specific tasks.
- Executed independently of the main code.
- Typically have prologues at the start to prepare the stack and registers.
- Epilogues at the end to restore the original state before the function call.

Windows API:

- Set of functions and procedures by Microsoft Windows.
- Interface for software applications to interact with Windows OS.
- Provides functions for various tasks:
  - Creating windows.
  - Managing user interfaces.
  - Handling input devices.
  - File and directory operations.
  - Networking.
- Standardizes communication between applications and Windows.
- Used for developing Windows software in various programming languages.

Windows Operating System has an API to interact with it as any other system.

Windows Operating System has two modes:

- User Mode
- Kernel Mode.
- All of user’s programs run on the user mode, then the Operating System (OS) takes the call from the user mode to kernel mode as per the Windows OS architecture to be performed on the low level and the results to be reverted back to the user’s program in the user mode.
- Any Windows program that makes activities on the system, is making Windows API calls at some point of time to interact with the operating system.
- These activities can be File System activities, Network activities, Information gathering activities about the system, or any other activities.
- For example, File System activities can be reading, modifying, or deleting files.

Windows Registry:

- Central hierarchical database for system configuration.
- Stores information for users, applications, and hardware devices.
- Four main registry files: System, Software, Security, and SAM.
- Located at "C:\Windows\System32\config\".
- Structure is similar to file system directories.
- Contains forensically valuable information.

Registry Investigation:

- Investigating the registry requires extraction from the computer.
- Registry is constantly updated, not a simple copy-paste.
- Third-party software like FTK Imager or EnCase Forensic is used for extraction.
- FTK Imager is a widely used tool.
- Research shows methods to extract registry info from Windows CE memory and volatile memory (RAM).

Registry Structure:

- Organized in a tree structure.

- Configuration settings stored as keys.
- Updates based on hardware and software changes.
- In Windows XP, 2000, and 2003 (NT-based OS), registry files are in "Windows\System32\Config."
- Structure resembles Windows folders and files.
- Main folders are "Hives," containing subfolders called "Keys," which have configuration information.

Registry Terminology:

- Root Key: The registry has five top-level sections called root keys (HKEY/hive), each with a specific purpose.
- Subkey: Similar to a subfolder within a folder.
- Key: A folder in the registry that can contain additional folders or values. Both root keys and subkeys are keys.
- Value Entry: An ordered pair with a name and value.
- Value or Data: The actual data stored in a registry entry.

Main Registry Hives (Root Keys):

- HKEY_CLASSES_ROOT (HKCR): Stores information defining types.
- HKEY_USERS (HKU): Defines settings for the default user, new users, and current users.
- HKEY_CURRENT_USER (HKCU): Stores settings specific to the current user.
- HKEY_LOCAL_MACHINE (HKLM): Stores settings that are global to the local machine.
- HKEY_CURRENT_CONFIG (HKCC): Stores settings about the current hardware configuration, particularly differences from the standard configuration.

Registry Hive and Supporting Files:

- Each hive has supporting files:
  - HKEY_CURRENT_CONFIG: System, System.alt, System.log, System.sav.
  - HKEY_CURRENT_USER: Ntuser.dat, Ntuser.dat.log.
  - HKEY_LOCAL_MACHINE\SAM: Sam, Sam.log, Sam.sav.
  - HKEY_LOCAL_MACHINE\Security: Security, Security.log, Security.sav.

- HKEY_LOCAL_MACHINE\Software: Software, Software.log, Software.sav.
- HKEY_LOCAL_MACHINE\System: System, System.alt, System.log, System.sav.
- HKEY_USER\.DEFAULT: Default, Default.log, Default.sav.

HKEY_LOCAL_MACHINE Hive:

- Contains five main keys, each with subkeys holding configuration information.
  - HARDWARE
  - SAM (Security Accounts Manager)
  - SECURITY
  - SOFTWARE
  - SYSTEM

Registry Hive File Structure:

- Hive files are allocated in 4096-byte blocks, starting with a header (base block) and continuing with a series of hive bin blocks. Each hive bin (HBIN) is typically 4096 bytes.

Command and Control (C2) Attacks:

- A type of attack where a malicious server is used to control compromised machines over a network.
- C2 server receives desired payloads from compromised machines.

How C2 Attacks Work:

- Initial infection often occurs through:
  - Phishing emails or instant messages.
  - Malvertising.
  - Vulnerable web browser plugins.
  - Direct installation of malware (with physical access).
- Compromised machine establishes communication with C2 server, awaiting instructions.
- Commands from the C2 server typically lead to the installation of more malware, giving attackers full control.

- Can lead to the creation of a botnet, a network of compromised machines.

Risks of C2 Attacks:

- Data theft.
- Shutdown of machines or even entire networks.
- Repeated shutdowns and reboots.
- Malware or ransomware attacks.
- Distributed denial of service (DDoS) attacks using botnets.

Different C2 Architectures:

1. Centralized architecture: Single server manages all responses.
2. Peer-to-peer architecture: Infected computers act as nodes, passing messages to each other.
3. Random architecture: Uses random sources to send commands, making detection difficult.

Attack Flow of a C2 Attack:

- Initial infection through phishing, malvertising, or direct malware installation.
- Compromised system establishes a C2 channel.
- Infected system receives instructions from the C2 server, leading to further actions, potentially compromising more hosts.

Examples:

- Twitter, Facebook faced C2 attacks in 2013.

Defenses Against C2 Attacks:

- **For System Administrators:**
  - Provide security awareness training.
  - Monitor networks.
  - Use AI-based Intrusion Detection Systems (IDS).
  - Limit user permissions.

    - Implement Two-factor Authentication (2FA).
    - Use digital code-signing.

- **For Users:**
    - Avoid opening email attachments.
    - Do not click on email links (URLs).
    - Use a firewall.
    - Log out and reboot your computer.
    - Utilize strong, complex passwords.
    - Employ antivirus software.
    - Keep your operating system updated.
    - Avoid clicking on pop-ups.
    - Be cautious and do not ignore warnings from your browser.

PowerShell:

- Built-in command shell in Windows.
- Offers flexibility and functionality for managing Windows systems.
- Often exploited by attackers due to its in-memory execution and presence on all Windows systems.

Why Attackers Use PowerShell:

- Signed by Microsoft and present on most Windows systems, making it ideal for stealth.
- Allows encoding input as commands, avoiding file drops for obfuscation.
- Used throughout the attack lifecycle, from initial infection to post-exploitation.
- Offers bitwise and string operations for high obfuscation.
- Facilitates advanced actions like reflective DLL injection and shellcode execution.

How Attackers Use PowerShell:

- Command and Control (C2) Communication.
- Credential Theft.

- Lateral Movement.
- Execution of Fileless Malware.
- Data Manipulation.

Living Off the Land (LOTL):

- PowerShell attacks are a type of LOTL attack.
- LOTL techniques use legitimate tools and functionalities on the target system to avoid traditional malware detection.

Types of PowerShell Attacks:

- Use in malicious macros.
- Disabling AMSI (Antimalware Scan Interface) to evade detection.
- Post-exploitation use of tools like Mimikatz.
- Encoded and compressed commands for obfuscation.

Defenses Against PowerShell Attacks:

- Restrict PowerShell usage to authorized users and trusted scripts.
- Implement PowerShell logging and monitoring for suspicious activity.
- Use Endpoint Detection and Response (EDR) solutions to detect malicious PowerShell activity.
- Apply software updates and patches to remediate vulnerabilities.
- Implement network segmentation to limit lateral movement.
- Use threat intelligence for early detection and prevention of attacks.

Malware writers use DLLs in three ways:

- To store malicious code
- By using Windows DLLs
- By using third-party DLLs

Exceptions:

- Handle events outside normal execution flow.

- Mostly caused by errors, e.g., division by zero.

Handling:

- Exception triggers a transfer to a specific routine for resolution.

Sources:

- Some exceptions, like division by zero, from hardware.
- Others, like invalid memory access, from the OS.

Explicit Raising:

- "RaiseException" call allows explicit exception generation in code.

Persistence Mechanisms:

- Persistence refers to malware's ability to remain active even after system reboots.
- Allows malware to evade detection and continue causing harm.
- Goals include data theft, ransom demands, and further attacks on other systems.

Common Malware Persistence Mechanisms:

- Modifying the registry.
- Creating scheduled tasks.
- Installing as a service.
- Using rootkits to hide presence.

Registry Run Keys:

- Modify registry keys to run automatically at system startup.
- Achieve persistence by editing keys in specific locations in the registry.

Startup Folders:

- Use folders to launch programs when the user logs in.
- Malware can place copies in startup folders for automatic execution.

Scheduled Tasks:

- Use Windows feature to automate program/script execution.

- Malware creates scheduled tasks for persistence.

Windows Services Registry:

- Malware can install itself as a service for background execution.

BootExecute Key:

- Part of the system boot process.
- Executed by Session Manager Subsystem (smss.exe).
- Contains values that launch at boot, typically "autocheck autochk*".

Persistence in the Registry:

- Malware edits registry keys for persistence, both at the user and admin levels.

System Boot and BootExecute Key:

- System boot involves processes, including the Session Manager Subsystem.
- BootExecute key in the registry used to specify processes that launch at boot. Malware may add itself to this key for persistence.

Browser Helper Objects (BHO):

- BHO is a dynamic-link library (DLL) file loaded by the browser during startup.
- It allows malware to run its payload each time the browser starts, ensuring persistence.
- Some BHOs are legitimate and provide additional browser functionality.

AppInit DLLs:

- DLLs loaded by Windows when a user logs in before other applications & can make malware difficult to remove.
- The OS reads a list of DLLs to load from the Windows Registry.
- Malware can establish persistence by adding a malicious DLL to the AppInit_DLLs key, running its payload at login.

- AppInit_DLLs are DLLs loaded into every process that loads User32.dll.
- Registry location: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Windows.

DLL Search Order Hijacking:

- Involves placing a malicious DLL with the same name as a legitimate one in a location searched by the OS before the legitimate location.
- When a program tries to load the DLL, the OS loads the malicious one, allowing malware to run its payload and establish persistence.
- Can also compromise legitimate programs by replacing their DLLs with malicious ones.

Rootkits:

- Rootkits hide other malware on a system.
- They conceal files, processes, and system components to evade detection.
- This makes it challenging for security software to detect and remove malware.
- Rootkits can persistently harm systems, steal data, or launch further attacks.

Bootkits:

- Bootkits infect the master boot record (MBR) or bootloader components.
- They execute before the operating system starts during boot.
- Bootkits are hard to remove as they automatically run every time the system boots.

Fileless Malware:

- Fileless malware operates in system memory without creating physical files on the disk.
- It avoids traditional antivirus detection.
- Often utilizes the registry for persistence or configuration data.
- Modifies registry entries to achieve automatic execution at system startup or user login.

Persistence in Windows Registry Entry:

- Common registry keys for achieving persistence include:
  - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
  - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
  - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
  - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
- Malware modifies these registry keys to ensure it runs automatically.

Privilege Escalation:

- Privilege escalation is required when a user isn't running with administrator rights.
- Common attacks include exploits, zero-day attacks, and privilege-escalation attacks.
- Recognizing privilege escalation techniques is important for analysts.
- Malware can abuse privileges such as SeDebugPrivilege to gain full system access.

SeDebugPrivilege:

- SeDebugPrivilege is a debugging privilege typically granted to local administrator accounts.
- It's used to gain access to system-level functions.
- Adjusting the access token with AdjustTokenPrivileges can enable SeDebugPrivilege.
- Malware exploits this privilege for full system access.

Self Defending Malware

Identifying Packed Programs:

- Few imports, especially if limited to LoadLibrary and GetProcAddress.
- Limited recognized code when opened in IDA Pro.
- OllyDbg may issue a warning about potential packing.
- Section names that indicate a specific packer (e.g., UPX0).
- Abnormal section sizes, like a .text section with Size of Raw Data as 0 and Virtual Size as nonzero.

Entropy Calculation:

- A measure of disorder in a system or program.
- No standard mathematical formula for entropy, but various measures exist for digital data.
- Compressed or encrypted data exhibits high entropy, resembling random data.
- Uncompressed or unencrypted executables have lower entropy.

Automated Detection Tools:

- Tools like Mandiant Red Curtain use heuristics, including entropy, to detect packed programs.
- Red Curtain calculates a threat score based on various measures and can scan a filesystem for suspected packed binaries.

Packing Techniques:

- Packers take an executable file as input and produce an executable file as output.
- Compression is a common technique used by packers to reduce the size of the original executable.
- Advanced packers may encrypt the original executable and employ anti-reverse-engineering methods like anti-disassembly, anti-debugging, or anti-VM.
- Packers can compress the entire executable, including data and resource sections, or only pack the code and data sections.
- Import information must be preserved in the packed program to maintain functionality.

Unpacking Stub:
The unpacking stub has three main tasks:

1. Unpack the original executable into memory.
2. Resolve all imports of the original executable.
3. Transfer execution to the original entry point (OEP).

Tail Jump:
Once the unpacking stub completes its tasks, it transfers execution to the OEP, often through a jump instruction. Some malicious packers use techniques like ret, call instructions, or OS functions (e.g., NtContinue) to obscure this process.

Process of Unpacking:

- **Original Executable:** Shows the original executable with its header, sections, and the starting point set to the OEP.
- **Packed Executable (on Disk):** Shows the packed executable on disk, displaying only the new header, unpacking stub, and packed original code.
- **Packed Executable (in Memory):** Displays the packed executable as loaded into memory after the unpacking stub unpacks the original code. .text and .data sections become visible, but the starting point still points to the unpacking stub, and the import table may not be valid at this stage.
- **Fully Unpacked Executable:** Shows the fully unpacked executable with a reconstructed import table and the starting point pointing to the OEP.

Anti-Disassembly:

- Anti-disassembly employs specially crafted code or data to confuse disassembly analysis tools.
- It's used by malware authors to delay or prevent the analysis of malicious code.
- This technique leverages the assumptions and limitations of disassemblers, tricking them into showing incorrect program listings.
- Different disassemblers may produce varying results, leading to inaccuracies and obscured program functionality.

Anti-Disassembly Techniques:

- **Objective:** Malware authors employ anti-disassembly techniques to create confusion and increase the level of skill required for reverse engineering.
- **Exploiting Disassembler Assumptions:** They take advantage of the assumptions and limitations of disassemblers, which typically represent each byte of a program as part of one instruction at a time.
- **Tricking Disassemblers:** Malware authors craft sequences of instructions that deceive disassemblers into displaying a list of instructions that differ from those that will actually be executed during runtime.
- **Disassembly Variability:** These techniques lead to variability in the disassembly output, resulting in different sets of instructions being presented for the same set of bytes.
- **Concealing Valid Instructions:** By disassembling at the wrong offset or manipulating the order of instructions, valid instructions can be hidden from view, making it challenging for analysts to discern the actual functionality of the code.

- **Increased Reverse Engineering Difficulty:** Anti-disassembly techniques increase the complexity of analyzing the code, requiring reverse engineers to invest extra effort and expertise in understanding the program's behavior.

Disassembly Strategies:

- *Linear Disassembly:*
  - Linear disassembly iterates over a code block and disassembles one instruction at a time in a linear, sequential manner. This approach is straightforward and is used in debugger writing tutorials.
  - It relies on the size of the disassembled instruction to determine the next byte to disassemble.
- *Flow-Oriented Disassembly:*
  - Flow-oriented disassembly is employed by most commercial disassemblers like IDA Pro.
  - Unlike linear disassembly, it does not blindly iterate over a buffer but examines each instruction to build a list of locations to disassemble.

Linear vs. Flow-Oriented Disassembly:

- Linear Disassembly: Produces an inaccurate set of instructions and misses important information, often due to misrepresenting instruction sequences.
- Flow-Oriented Disassembly: Provides more accurate results, as its logic better mirrors the real program's execution flow.

Disassembly is a complex process, and anti-disassembly techniques exploit the intricacies of disassemblers to make reverse engineering more challenging.

- **Anti-Debugging:**
  - *Purpose:* Anti-debugging techniques are used by malware to detect when they are being analyzed or debugged, aiming to hinder the efforts of malware analysts.
  - *Objectives:* Malware authors employ anti-debugging methods to recognize debugger presence or thwart debugging attempts by modifying their execution path or causing crashes.

- **Windows Debugger Detection:**
  - *Debugger Presence Detection Methods:*
    - Malware uses various techniques to detect the presence of a debugger:
      - **IsDebuggerPresent:** This API function checks the Process Environment Block (PEB) structure for the IsDebugged field to determine debugger presence.
      - **CheckRemoteDebuggerPresent:** Similar to IsDebuggerPresent, but it checks for a debugger in another process on the local machine.
      - **NtQueryInformationProcess:** A native API function that retrieves process information, including debugging status. It uses the ProcessDebugPort parameter to determine if the process is being debugged.
      - **OutputDebugString:** Used to send a string to a debugger for display. It can be employed to detect a debugger's presence.

Anti-debugging is a common tactic employed by malware to impede analysis and make it more challenging for analysts to understand the malware's behavior. Malware can take various actions once it detects a debugger, including modifying its code execution or causing crashes to disrupt analysis efforts.

- **Anti-Virtual Machine (Anti-VM) Techniques:**
  - Malware may use anti-VM techniques to detect if it is running inside a virtual machine. If a VM is detected, malware can behave differently or avoid execution, complicating the analysis process.
  - Anti-VM techniques are commonly used in widely deployed malware like bots, scareware, and spyware, as these often target average users' machines, which are less likely to run in a virtual environment. Honeypots frequently use virtual machines, making them susceptible to anti-VM detection.

- **VMware Artifacts:**
  - The VMware virtualization environment leaves several artifacts on the system, particularly when VMware Tools is installed. Malware can use these artifacts found in the filesystem, registry, and process listing to detect VMware.
  - For instance, the presence of processes like VMwareService.exe, VMwareTray.exe, and VMwareUser.exe can indicate a VMware environment.

- Additionally, information in the registry and the installation directory (e.g., C:\Program Files\VMware\VMware Tools) may reveal VMware artifacts. These artifacts contain data about the virtual hardware and mouse, among other things.

- **Bypassing VMware Artifact Detection:**
  - Defeating malware that searches for VMware artifacts generally involves two steps: identifying the detection method and patching it.
  - For example, if malware scans for specific VMware-related strings in process listings, you can:
    - Patch the malware binary during debugging to prevent it from taking the specified action.
    - Use a hex editor to alter the VMware-related string (e.g., "VMwareTray.exe") to an invalid value to fail the comparison.
    - Uninstall VMware Tools from the system to remove processes like VMwareTray.exe, preventing detection based on their presence.

Thanks for reading my notes! I hope it was helpful in your learning curve. For more such content follow me on my GitHub and Twitter :)

GitHub:

cyph3rryx - Overview

21 y/o 😁 • Cybersecurity Student💫 • CTF & Bug Bounty 👾• Security Researcher 🪲 • Screenwriter 😉• Nerd 😎 • Top 3% on TryHackMe (New Ranking System) 😌 - cyph3rryx

https://github.com/cyph3rryx

Twitter:

Ryx (@PadhiyarRushi) / X

21 y/o • Cybersecurity Student • CTF & Bug Bounty • Security Researcher • Screenwriter • Graphic Designer • In Top 3% @RealTryHackMe

https://twitter.com/PadhiyarRushi