Unit 1: Introduction to Malware and Reverse Engineering

1. Need for Reverse Engineering and Malware Analysis

- Understanding malware behavior helps detect, respond, and defend against cyber threats.
- It improves security strategies and strengthens incident response.

2. What is Malware?

- Malware refers to malicious software designed to harm or exploit systems.
- It spreads through emails, websites, USB drives, and exploits user trust.

3. Uses of Malware

a) Financial Gain

- Attackers steal data, demand ransoms, or engage in fraud for monetary rewards.

b) Psychological Urges

- Some individuals create malware for thrill, challenge, or to “beat the system.”

c) Access Private Data

- Malware helps attackers access confidential information like personal, business, and financial data.

4. Typical Purposes of Malware Attacks

a) Backdoor Access

- Creates hidden ways for attackers to remotely control infected systems.

b) Denial-of-Service (DoS) Attacks

- Overwhelms servers to make them crash or unavailable.

c) Vandalism

- Defaces websites or damages digital assets to make a statement.

d) Resource Theft

- Steals network bandwidth, computing power, or storage from victims.

e) Information Theft

- Steals sensitive information like credit card details, passwords, etc.

5. Malware Analysis

- It involves dissecting malware to understand how it works and how to defend against it.
- Helps in categorizing threats, improving response, and enriching threat intelligence.

6. Types of Malware

a) Viruses

- Self-replicating programs needing human action to spread; may delete or corrupt data.

b) Worms

- Self-replicating but spreads without user action, often via internet vulnerabilities.

c) Trojan Horses

- Malware disguised as legitimate software; secretly harms once installed.

d) Backdoors

- Provides secret entry points for attackers without user consent.

e) Mobile Code

- Active web components like Java applets that can be misused for attacks.

f) Adware

- Shows unwanted ads and collects user data, often slowing down systems.

g) Sticky Software

- Malware that's very hard to remove, often reinstalls itself after deletion.

7. Future Malware Trends

a) BIOS/Firmware Malware

- Malware at hardware-level; hard to detect and remove using normal antivirus.

b) Information-Stealing Worms

- Worms using encryption to hold stolen data hostage for ransom.

8. Reversing Malware

- Reverse engineering malware reveals its behavior and weaknesses.
- It's crucial for developing effective defenses and clean-up strategies.

9. Static and Dynamic Analysis

a) Static Analysis

- Examining malware code without running it; involves disassembling and signature scanning.

b) Dynamic Analysis

- Running malware in a sandbox environment to monitor real-time behavior.

10. Steps in Reverse Engineering

- Obtain malware sample ➔ Disassemble ➔ Analyze code ➔ Create sandbox ➔ Run and observe ➔ Report findings.

11. Static Malware Analysis Tools

a) BinText

- Extracts readable strings from executables revealing registry keys and commands.

b) IDA Pro

- Converts executable code to human-readable assembly code.

c) UPX

- Handles packing and unpacking of compressed executables.

d) Proc Dump

- Dumps active memory contents to a file for offline analysis.

e) OllyDbg

- A debugger allowing live inspection of running processes.

12. Dynamic Malware Analysis Tools

a) Process Explorer

- Displays active processes and their details.

b) FileMon

- Monitors file system activity to detect malware file operations.

c) RegMon

- Monitors changes made to the Windows registry.

d) RegShot

- Takes before-and-after snapshots of registry changes.

e) TCPView

- Shows all open TCP/UDP connections and the processes using them.

f) TDIMon

- Logs network traffic at the transport driver interface level.

g) Ethereal (Wireshark)

- Captures and analyzes network packets for malicious activity.

13. Moral of the Story

- Reverse engineering and malware analysis are distinct but complementary skills.
- Both are vital to defend against constantly evolving cyber threats.

14. Conclusions

- Developers must code securely and users must stay educated about malware threats.
- Hands-on practice with real malware samples strengthens understanding and defense capabilities.

Unit 5: Examining Self-Defending Malware

1. Overview

- Malware increasingly uses self-defense techniques to evade detection.
- Anti-debugging and sandbox evasion make analysis and response harder.

2. Anti-Debugging

- Malware detects debugging attempts and alters behavior to avoid reverse engineering.
- Delays or prevents cybersecurity experts from analyzing the code.

3. Best Techniques of Anti-Debugging

a) Code Modification Detection

- Identifies changes made by debuggers to the running code.

b) Adding Exceptions in Code

- Unhandled exceptions indicate the presence of a debugger.

c) Self-Debugging

- Malware attaches to itself to detect if another debugger is active.

d) System API Calls

- Calls like `IsDebuggerPresent` check if a debugger is attached.

e) Exploiting Debugger Bugs

- Sends inputs that exploit vulnerabilities in popular debuggers.

f) Hardware Breakpoint Detection

- Checks for unusual breakpoints set by debugging tools.

g) Timing-Based Detection

- Detects debugging by monitoring delays or execution timing anomalies.

h) Library Checking

- Detects debugger-specific libraries loaded into the application.

i) Mixing Techniques

- Combines several anti-debugging methods for stronger protection.

4. Special Anti-Debugging Functions

a) IsDebuggerPresent

- Checks if the program is running under a debugger.

b) PEB (Process Environment Block)

- Inspects internal process structures to detect debuggers.

c) TLS Callback

- Early execution points in programs, good for sneaky debugger checks.

d) NtGlobalFlag

- Used to detect debugged processes through system-level flags.

5. Process Hollowing

- Malware replaces the memory of legitimate processes with malicious code.
- Common methods involve suspending a process, hollowing it, injecting code, and resuming.

6. Examples of Malware Using Process Hollowing

- Agent Tesla, Astaroth, Azorult, BADNEWS, Bandook, Bazar, BBSRAT, Clambling, Cobalt Strike.
- These malware use hollowing to blend in and evade security tools.

7. Attack Surface Reduction (ASR) Rules

- Microsoft Defender rules reduce points of attack by controlling script executions and driver installations.
- Helps harden systems against exploitation attempts.

8. Sandboxing and Malware Evasion

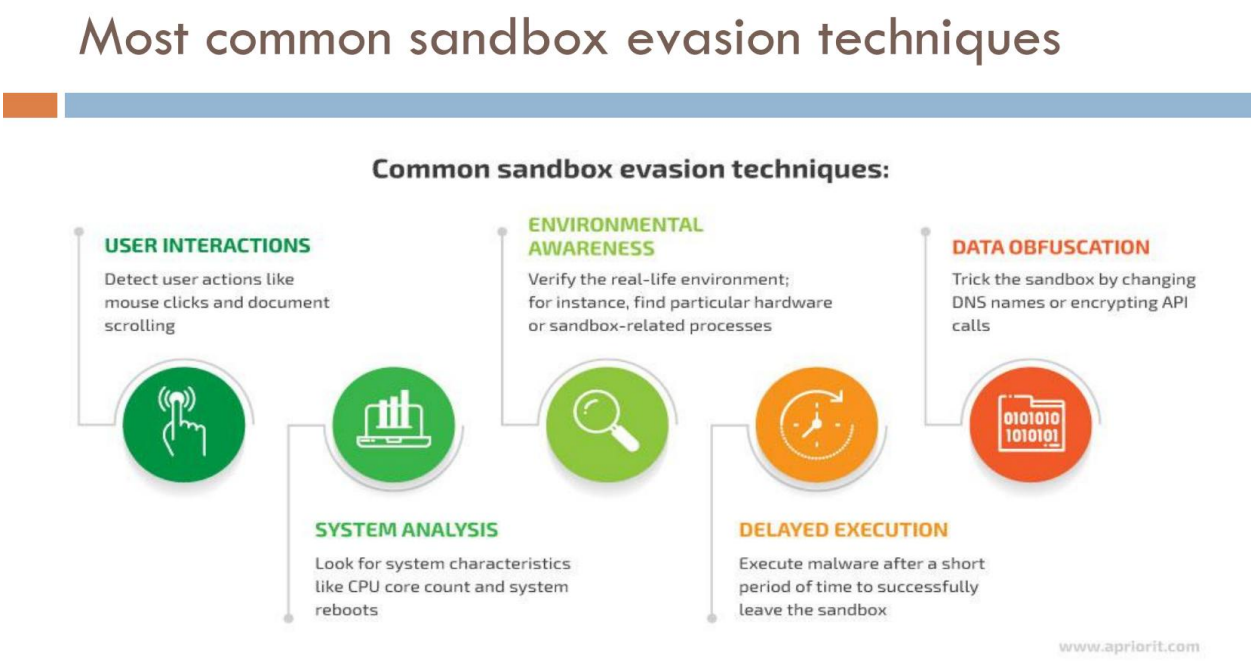
- Malware detects sandbox environments and delays or hides its execution.
- Makes automated malware analysis less effective.

9. What is Sandbox-Evading Malware?

- Malware that remains inactive inside sandboxes and activates only in real environments.
- Uses environment checks and delayed activation to escape detection.

10. Examples of Sandbox-Evading Malware

- Locky ransomware, RogueRobin Trojan, HAWKBALL backdoor, OSX_OCEANLOTUS.D malware.
- These malware have sophisticated evasion techniques.



11. Most Common Sandbox Evasion Techniques

a) Detecting User Interactions

- Waits for real human actions like scrolling or mouse movement before activating.

b) Detecting System Characteristics

- Looks for clues like CPU cores, digital signatures, or installed software to identify sandboxes.

c) Environmental Awareness

- Detects sandbox-specific services, filenames, or processes to stay dormant.

12. Other Sandbox Evasion Techniques

a) Timing-Based Techniques

- Malware uses extended sleep, logic bombs, or stalling to wait out sandbox analysis.

b) Obfuscating Internal Data

- Encrypts its code or communications to avoid analysis.

13. How to Detect Sandbox-Evading Malware

a) Dynamically Change Sleep Duration

- Trick malware by altering time settings inside the sandbox.

b) Simulate Human Interactions

- Emulate realistic user activity to activate hidden malware.

c) Add Real Environmental Artifacts

- Mimic real hardware and system artifacts to fool the malware.

d) Perform Static and Dynamic Analysis

- Combine both code and behavior analysis for more thorough detection.

e) Use Fingerprint Analysis

- Look for common malware patterns or evasion tactics inside files.

f) Behavior-Based Analysis

- Analyze malware behavior to detect sandbox-aware characteristics.

g) Customize Sandbox Environments

- Use multiple varied sandbox setups to catch evasion attempts.

h) Add Kernel Analysis

- Detect malware that tries to move into low-level system areas like drivers.

i) Implement Machine Learning

- Use AI models to spot subtle evasion attempts early.

j) Use Content Disarm and Reconstruction (CDR)

- Remove active contents from files to neutralize embedded threats.