Malware Analysis Project

Name : Lavanya Chilukamari

Intern ID : 308

Defination : Malware analysis is the process of studying, identifying, and understanding malicious software (malware) to learn how it works, what it does, and how to detect, prevent, or remove it.

Purpose of Malware Analysis

- Identify the type of malware (virus, trojan, ransomware, etc.)
- Understand its behavior (what files it modifies, network activity, processes it spawns)
- Create detection signatures (YARA, antivirus rules)
- Develop countermeasures (patching, removal tools)

What is a Malware Analysis Fingerprint?

A malware analysis fingerprint is a unique set of characteristics or indicators left behind by a specific malware sample or family, which can be used to identify, detect, or classify that malware.

Finding malware fingerprint using memory analysis

1. Acquire Memory Dump

Use trusted tools to capture a snapshot of RAM:

- Windows:
  - DumpIt (FireEye)
  - FTK Imager
  - WinPMEM (Google Rekall)
- Linux:
  - LiME (Linux Memory Extractor)
- Mac:
  - OSXPmem

2. Load Memory Dump into Analysis Tool

Use a forensic memory analysis framework:

- Volatility / Volatility3
- Redline (FireEye – more GUI based)

3. Identify Indicators of Compromise (IOCs)

Check for:

- Processes with no parent (PPID 0) or odd names (e.g., svch0st.exe)
- Memory regions marked RWX (read-write-execute)
- Suspicious strings: hardcoded IPs/domains, base64 blobs, PowerShell scripts
- Unusual network activity (to foreign IPs, uncommon ports)
- Persistence mechanisms (e.g., registry run keys, services)
- Packaged malware (UPX-packed, etc.)

4. Correlate with Threat Intelligence

Match findings against:

- Known malware hashes (VirusTotal)
- IPs/domains (AbuseIPDB, AlienVault OTX)
- MITRE ATT&CK tactics/techniques

Example: Finding Injected Code

Look for:

- Injected memory regions
- Suspicious process names (e.g. rundll32.exe with unknown DLL)
- Non-PE (non-standard) code in executable memory

Fingerprint of malware analysis example :

Code :

```
import subprocess

def run_volatility_analysis(memory_file, profile):
    plugins = [
        "windows.pslist",         # List running processes
        "windows.malfind",        # Detect injected code (DLL injection, shellcode)
        "windows.cmdline",        # Get process command line arguments
        "windows.driverscan",     # Detect suspicious drivers
        "windows.ssdt",           # Check System Service Descriptor Table hooks
        "windows.modules",        # Check loaded kernel modules
        "windows.strings"         # Search suspicious strings
    ]

    for plugin in plugins:
        print(f"\n[+] Running plugin: {plugin}")
        command = [
            "volatility3",
            "-f", memory_file,
            "--plugin-dirs", ".",
            plugin
        ]
```

```
        try:
            result = subprocess.run(command, capture_output=True, text=True)
            output = result.stdout
            print(output)
        except Exception as e:
            print(f"[-] Error running plugin {plugin}: {e}")

# Example usage
if __name__ == "__main__":
    memory_image = "memory.raw"  # Replace with your memory dump file
    run_volatility_analysis(memory_image, "Win10x64")
```

Output :

```
Process: svchost.exe Pid: 1234
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Malware-like behavior: Shellcode detected
Dumped to: svchost.0x7ffe0000.0x1000.dmp

Process: notepad.exe Pid: 2345
Vad Tag: VadS Protection: PAGE_EXECUTE_READ
No suspicious injection
```

WinHex Tool in Malware Analysis

WinHex is a powerful hexadecimal editor widely used in digital forensics, including malware analysis. It allows low-level inspection of files, memory dumps, disks, and more. Here's how WinHex can help in malware analysis:

What is WinHex?

WinHex is a universal hexadecimal editor that can:

- Read and modify binary files
- View and analyze RAM, disk sectors, and image files
- Recover deleted files or data
- Identify hidden or obfuscated content (e.g., embedded malware)

Common Steps Using WinHex

1. Open a Memory Dump or File
   - File → Open → Choose .raw, .img, .dmp, .exe, etc.
2. Analyze Header
   - Check for abnormal PE headers, fake extensions (e.g., .jpg with .exe content)
   - Verify if file is packed, encrypted, or malformed
3. Search for Suspicious Strings

- Use Search → Find Text or Find Hex Values to find:
    - "cmd.exe"
    - "powershell"
    - "base64"
    - Known malware signatures
    - Embedded shellcode (e.g., calc.exe, IPs, URLs)

4. View Entropy Levels
    - Use Tools → Specialist → File Tools → Entropy to detect compressed/encrypted (possibly malicious) data.

5. Recover Injected/Hidden Code
    - Manually carve out binary blobs
    - Save selected block → Open in PE Studio or IDA for further analysis

Example: Detecting Shellcode in Memory Dump

1. Open memory dump in WinHex
2. Search for suspicious byte patterns or strings
3. Identify memory region marked with shellcode-like instructions (e.g., lots of NOP, suspicious opcodes)
4. Mark the region, extract it
5. Analyze it in x64dbg, IDA, or Ghidra

Example of Tools in WinHex :

Step 1: Load Memory Dump in WinHex

- Open WinHex.
- Use File → Open and load memory.dmp.
- Switch to hex view to see raw memory contents.

Step 2: Search for Known Malware Fingerprint (e.g., String)

- Suppose you're looking for a malware signature:
  `XOR key = 0x90`, string `"cmd.exe"` (a common indicator of code injection or process hollowing).

WinHex Script Example :

```
// WinHex script to search for 'cmd.exe' string
MessageBox("Start scanning memory for cmd.exe...");

SetCursor(0);             // Start from beginning
SearchASCII("cmd.exe");   // Search for ASCII string

If Found()
    MessageBox("Found 'cmd.exe' at offset: " + ToHex(GetCursor()));
Else
    MessageBox("String not found.");
EndIf
```

Ouput :

```
MessageBox: Found 'cmd.exe' at offset: 0x01A3F002
```