GUIDE TO MALWARE REVERSE ENGINEERING AND INCIDENT RESPONSE WITH EXAMPLES AND SIMULATIONS

BY IZZMIER IZZUDDIN

Table of Contents

REVERSE ENGINEERING

Key Goals Of Reverse Engineering

1. **Analyse Malware Behaviour**
    - Reverse engineering allows security analysts to break down malware samples and understand their functionality. By examining how the malware behaves (e.g., which processes it starts, how it propagates, how it communicates with Command & Control servers), you can determine the full extent of its capabilities.
2. **Determine Vulnerabilities Being Exploited**
    - By deconstructing a piece of malware, analysts can identify the specific vulnerabilities the malware exploits. This helps not only in understanding the threat but also in patching these vulnerabilities to prevent future attacks.
3. **Develop Detection Signatures and Heuristics**
    - Once you know how the malware works, you can create detection signatures for Intrusion Detection Systems (IDS), antivirus programs and other security tools. This could be based on specific code patterns, behaviour during execution or known indicators of compromise (IoCs).
4. **Strengthen Defence Mechanisms**
    - Reverse engineering insights help improve the entire cybersecurity posture. By learning how the malware bypassed defences, analysts can enhance existing controls (e.g., antivirus engines, firewall rules, application whitelisting) to better protect against similar threats in the future.

Tools For Reverse Engineering

1. **Static Analysis Tools**
    - **IDA Pro**
      A powerful disassembler that converts binary code into assembly code, allowing detailed exploration of the malware's structure and functions. Useful for identifying function calls, variables and control flow.
    - **Ghidra**
      An open-source tool created by the NSA for reverse engineering, it offers similar capabilities to IDA Pro, including decompilation into high-level pseudo-code.
    - **PEiD**
      A tool for identifying the packer or compiler used to create the executable, helping to determine if the malware is packed or obfuscated.

2. **Dynamic Analysis Tools**

   - **Cuckoo Sandbox**
     An automated malware analysis system that executes malware in a virtual environment and captures its behaviour, including network traffic, file system changes and registry modifications.

   - **OllyDbg**
     A debugger that lets you step through the malware's execution, monitor its behaviour and manipulate its flow to analyse its real-time actions.

   - **Process Monitor (Procmon)**
     A Windows-based tool that logs system-level events like file, registry and process activity, providing insight into malware's real-time behaviour.

3. **Memory Forensics Tools**

   - **Volatility:** A memory forensics framework that analyses memory dumps for injected code, running processes and other artifacts left by malware in RAM.

Reverse Engineering Workflow

1. **Static Analysis**

   - **Identify Basic Information**
     Examine the file header, imports and strings to understand what the malware might do.

   - **Disassemble the Code**
     Use IDA Pro or Ghidra to disassemble the code and explore its logic. Look for common indicators like network connections, process injection and obfuscation techniques.

   - **Unpacking**
     If the malware is packed, unpack it to reveal the true code. Tools like UPX can help with this, but sometimes manual unpacking is needed using debuggers.

2. **Dynamic Analysis**

   - **Behaviour Analysis**
     Execute the malware in a sandbox (like Cuckoo) to see what actions it performs. Capture network traffic using tools like Wireshark and observe file and registry changes with Procmon.

   - **Debugger Analysis**
     Step through the malware code in OllyDbg or x64dbg. This can reveal hidden functionality like anti-analysis techniques (e.g., checking for debuggers or virtual environments).

3. **Advanced Memory Analysis**

    - **Dump Memory**
      Use memory forensics to capture a memory dump of the infected system and analyse it with tools like Volatility. This helps in identifying injected processes or malware that only runs in memory.

    - **Decrypt In-Memory Artifacts**
      Some malware encrypts data or only loads sensitive components into memory. Extract and decrypt these components to fully analyse the threat.

4. **Report Findings**

    - Summarise key findings, including discovered IoCs (e.g., file hashes, IP addresses, registry keys) and document how the malware operates. Share insights with security teams for remediation and future defence measures.

Example: Reverse Engineering A Ransomware Variant

1. **Static Analysis:**

    - Initial scan reveals the binary is packed with a custom packer.

    - After unpacking, we identify imports for CreateProcess, CryptEncrypt and WriteFile indicating encryption behaviour likely tied to the ransomware's payload.

2. **Dynamic Analysis:**

    - In a controlled sandbox, the ransomware executes, encrypting targeted file types (e.g., .doc, .jpg). Network traffic shows outbound connections to a TOR-based C2 server, possibly for key exchange or status updates.

3. **Memory Analysis:**

    - Memory dump reveals a decrypted version of the ransomware’s key handling routine, exposing a potential flaw in the key exchange mechanism that might be exploitable for decrypting files.

4. **Resulting Signature:**

    - An IDS signature is created to detect specific outgoing TOR traffic patterns, along with file system monitoring rules to catch encryption activity.

STEPS IN MALWARE REVERSE ENGINEERING

1. **Initial Triage and Static Analysis**

   - **Hashing**
     Calculate the hash (MD5, SHA-256) of the malware sample to determine if it's been seen before by checking threat intelligence sources.

   - **Metadata Extraction**
     Analyse the malware file's metadata (file size, type, timestamps, etc.) without executing it. Tools like *PEStudio* or *ExifTool* are helpful here.

   - **Strings Analysis**
     Extract readable strings from the malware binary to uncover hardcoded URLs, IP addresses, commands and function calls. Tools like *strings* or *BinText* are used for this.

   - **PE Analysis (for Windows malware)**
     Analyse the Portable Executable (PE) format for sections, imports and exported functions. Tools like *PE Explorer* or *CFF Explorer* assist in this task.

2. **Behavioural (Dynamic) Analysis**

   - **Sandbox Execution**
     Run the malware in an isolated environment (sandbox) to observe its behaviour, such as file system changes, network activity or registry modifications. Tools like *Cuckoo Sandbox* or *Any.Run* are commonly used.

   - **Network Traffic Analysis**
     Monitor the network communication of the malware to identify command-and-control (C2) servers, data exfiltration or further malware downloads. Tools like *Wireshark* or *Fiddler* are helpful here.

   - **Process Monitoring**
     Track the processes created or modified by the malware using tools like *Procmon* or *Process Explorer*.

3. **Code-Level (Static) Analysis**

   - **Disassembly**
     Break down the malware into assembly code using disassemblers like *IDA Pro* or *Ghidra* to analyse the malware's instructions without executing it.

   - **Debugging**
     Use debuggers like *x64dbg* or *OllyDbg* to execute the malware step-by-step in a controlled environment, allowing you to observe the code execution and find critical parts like decryption routines or key functions.

- **Deobfuscation and Unpacking**
  Many malwares use obfuscation techniques (e.g., packing) to hide their actual code. The reverse engineer needs to bypass these protections, often by dumping the unpacked code from memory.

4. **Advanced Techniques**

   - **Memory Forensics**
     Analysing memory dumps to uncover the malware's runtime behaviour and to extract encryption keys, unpacked payloads or other volatile information. Tools like *Volatility* or *Rekall* are used for this.

   - **Decompilation**
     Reverse engineer the compiled binary into a higher-level language like C or C++ for easier analysis. Tools like *Ghidra* or *Hex-Rays Decompiler* assist with decompiling.

EXAMPLES AND SIMULATIONS

Scenario 1: Malware Analysis

We received a suspicious file named invoice.doc.exe that was flagged during an incident in the network. Our objective is to reverse engineer the malware to understand its purpose, behaviour and produce IoCs.

Step 1: Static Analysis

1. **File Properties and Metadata**

- **File Name:** invoice.doc.exe
- **File Size:** 180 KB
- **File Type:** PE32 executable (GUI) Intel 80386, for MS Windows
- **MD5 Hash:** f2d4e3b7c0d3722b2c5b0f4a2a9f10c8

Actions Taken:

- **Hash Lookup**
  A quick search using the MD5 hash in online malware repositories yields no previous reports, suggesting it's a new or customised malware.
- **Metadata Extraction**
  The metadata reveals that the file was created using a known packer, *UPX*. This indicates that the malware is likely packed, a common technique used to evade detection.

2. **Strings Analysis**

Using the strings command, we extract readable strings from the binary. Key findings:

- **Interesting Strings:**
  - http://malicious-domain.com/connect
  - /home/user/data_exfil
  - AES Encryption
  - GetSystemInfo

These strings indicate possible malicious behaviour, including connecting to a remote server and data exfiltration.

3. **PE Structure Analysis**

Using *PEStudio*, we examine the Portable Executable structure:

- **Imported Functions:**
    - GetSystemInfo
    - CreateProcessA
    - WinExec
    - InternetOpenA
    - InternetConnectA

The presence of these imports suggests that the malware is attempting to collect system information and potentially execute other processes. The network-related functions (InternetOpenA, InternetConnectA) imply communication with a remote server.

Step 2: Dynamic Analysis

1. Sandbox Execution

We run the malware in a controlled environment using *Cuckoo Sandbox*. Below are the observed behaviours:

- **Process Creation**
  The malware spawns multiple processes, including cmd.exe and powershell.exe, likely for command execution.
- **File Creation**
  It creates a new file, C:\Users\Victim\temp\stolen_data.txt.
- **Network Activity**
  The malware attempts to connect to http://malicious-domain.com/connect on port 80 and sends an HTTP POST request containing encoded data.
- **Registry Modifications**
  The malware adds persistence by modifying the Run key in the Windows registry:
    - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\Malware: C:\Users\Victim\temp\malicious.exe

2. Network Traffic Analysis

We capture network traffic using *Wireshark*. Analysing the captured pcap file shows:

- **Outbound Traffic**
    - POST request to malicious-domain.com with the body containing base64-encoded strings.

    - The response from the server is a command to execute additional tasks on the infected system.

Decoding the base64 data reveals:

- **Stolen Information**
  Basic system information including the username, system hostname and local IP address.

Step 3: Code-Level Analysis

1. Unpacking the Malware

Since the malware is packed with UPX, we use the upx -d command to unpack the binary. After unpacking, we reanalyse the strings and imported functions.

2. Disassembly

Using *IDA Pro*, we disassemble the unpacked binary. Key observations include:

- **Main Function**
  The malware collects system information using GetSystemInfo() and sends it to the C2 server via HTTP POST.
- **Command Execution**
  The malware waits for commands from the server and executes them using CreateProcessA().
- **Encryption**
  It uses AES encryption to protect the data it exfiltrates, implemented through custom encryption routines located in a dedicated function.

Step 4: Memory Forensics

Using *Volatility*, we analyse a memory dump from the infected system. Key findings:

- **Unpacked Code**
  We locate the unpacked code in memory, allowing us to extract additional IoCs, such as encryption keys used by the malware.
- **Decryption Routine**
  By analysing memory, we can reconstruct the decrypted payload sent to the C2 server, which reveals detailed information about the infected system.

Indicators of Compromise (IoCs)

File Hashes:

- f2d4e3b7c0d3722b2c5b0f4a2a9f10c8 (malware executable)

File System:

- C:\Users\Victim\temp\stolen_data.txt
- C:\Users\Victim\temp\malicious.exe

Registry Keys:

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\Malware

Network Indicators:

- http://malicious-domain.com/connect
- IP Address: 192.168.1.100 (C2 Server)

Behavioural Indicators:

- Process creation of cmd.exe and powershell.exe
- Base64-encoded data in HTTP POST requests

Step 5: Final Report

Summary

The malware, identified as a custom Remote Access Trojan (RAT), collects system information and establishes persistence on the infected host. It communicates with a remote C2 server via HTTP and executes commands sent by the attacker. The RAT employs AES encryption to protect exfiltrated data, complicating analysis without the proper decryption routine.

Recommendations:

1. **Block C2 Domains and IPs:** Block all outbound traffic to malicious-domain.com and IP address 192.168.1.100.
2. **Quarantine Affected Systems:** Isolate any systems with IoCs to prevent further spread.
3. **Patch Vulnerabilities:** Ensure all systems are updated to prevent exploitation of known vulnerabilities.
4. **Enhance Detection Rules:** Update SIEM and IDS/IPS systems to detect base64-encoded POST requests and process anomalies like unauthorised cmd.exe or powershell.exe executions.
5. **Conduct Forensics:** Perform memory analysis on infected systems to extract encryption keys and recover exfiltrated data.

Scenario 2: Keylogger Analysis

A suspicious executable file named keylogger.exe has been found on a user's system. Our goal is to reverse engineer this malware to understand its capabilities and produce actionable Indicators of Compromise (IoCs).

Step 1: Static Analysis

1. **File Properties and Metadata**

- **File Name:** keylogger.exe
- **File Size:** 120 KB
- **File Type:** PE32 executable (console) Intel 80386, for MS Windows
- **MD5 Hash:** a1b2c3d4e5f67890123456789abcdefg

Actions Taken:

- **Hash Lookup**
  A search of the MD5 hash in online malware repositories yields no previous entries, indicating that it might be a new variant of a keylogger.
- **Metadata Extraction**
  No significant metadata reveals itself, indicating that the malware may be custom or compiled without any obfuscation tools.

2. **Strings Analysis**

We run the strings command to extract readable strings from the binary. Key findings:

- **Interesting Strings:**
  - logfile.txt
  - hookKeyboard
  - user32.dll
  - GetAsyncKeyState

The presence of hookKeyboard and GetAsyncKeyState suggests the malware is likely intercepting keystrokes by hooking keyboard input events.

3. **PE Structure Analysis**

Using *PEStudio*, we analyse the structure of the Portable Executable:

- **Imported Functions:**
  - GetAsyncKeyState

- SetWindowsHookExA
- WriteFile
- CreateFileA
- Send

The imports confirm that the malware is hooking keyboard events (SetWindowsHookExA) and capturing keystrokes (GetAsyncKeyState). The presence of WriteFile and CreateFileA implies that the malware is writing the captured keystrokes to a local file (logfile.txt) and the Send function suggests the possibility of exfiltrating this data.

Step 2: Dynamic Analysis

1. Sandbox Execution

We run the malware in an isolated environment using *Any.Run*. The following behaviours are observed:

- **File Creation**
  A file named logfile.txt is created in the directory C:\Users\Victim\Documents\.
- **Keystroke Logging**
  The malware captures every keystroke and appends it to logfile.txt. Typing "hello123" in a notepad results in the following entry in logfile.txt:
  - h,e,l,l,o,1,2,3
- **No Immediate Network Activity**
  Initially, no network traffic is observed. This suggests that the keylogger might be operating in stealth mode or exfiltrating data on a trigger event (e.g., reaching a file size threshold).

2. File System and Registry Monitoring

- **Persistence Mechanism**
  Monitoring the registry reveals that the malware adds a key under:
  - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\Keylogger: C:\Users\Victim\Documents\keylogger.exe
- **File Modification**
  The malware continually writes to logfile.txt as keystrokes are captured.

3. Network Traffic Analysis

Further execution of the malware reveals that after logfile.txt reaches a size of 5 KB, the malware initiates network activity:

- **Outbound Traffic**
  The keylogger attempts to send the contents of logfile.txt to http://malicious-keylogger.com/upload.
- **Packet Inspection**
  Using *Wireshark*, we observe an HTTP POST request with the content of logfile.txt being uploaded.

Step 3: Code-Level Analysis

1. Disassembly

Using *Ghidra*, we disassemble the binary and analyse the key functions:

- **Keylogger Hook Function**
  The malware hooks into Windows using SetWindowsHookExA to intercept keyboard events. The captured keystrokes are stored in memory and written to a file periodically.
- **Data Exfiltration Routine**
  The function responsible for sending the logfile.txt file over the network uses a basic HTTP POST request. We find that this routine is triggered whenever the log file reaches a certain size (5 KB).

2. Debugging

Using *x64dbg*, we run the malware in a controlled environment to observe the specific behaviour in real time. We find the following:

- **Logging and Triggering**
  The malware collects keystrokes and periodically checks the size of logfile.txt. Once the size exceeds the 5 KB threshold, it triggers the data exfiltration routine.

Step 4: Memory Forensics

Using *Volatility*, we analyse a memory dump from the infected machine to gather more information:

- **Unpacked Code**
  We locate the keylogging functionality within the memory dump and observe the stolen keystrokes in plaintext form.
- **Configuration Data**
  We extract configuration details that reveal the C2 server (malicious-keylogger.com) and additional parameters such as the file size threshold for triggering the upload.

Indicators of Compromise (IoCs)

File Hashes:

- a1b2c3d4e5f67890123456789abcdefg (malware executable)

File System:

- C:\Users\Victim\Documents\logfile.txt
- C:\Users\Victim\Documents\keylogger.exe

Registry Keys:

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\Keylogger

Network Indicators:

- http://malicious-keylogger.com/upload
- **IP Address:** 203.0.113.45 (C2 Server)

Behavioural Indicators:

- File creation and modification of logfile.txt
- Periodic network POST requests to exfiltrate logged keystrokes

Step 5: Final Report

Summary

The malware in question is a keylogger that hooks into Windows API functions to capture keystrokes. It writes the intercepted keystrokes to a local file (logfile.txt) and then exfiltrates this file to a remote C2 server (malicious-keylogger.com) after reaching a certain file size. The keylogger also establishes persistence by adding itself to the Windows registry.

Recommendations:

1. **Block C2 Domains and IPs:** Block all outbound traffic to malicious-keylogger.com and IP address 203.0.113.45.
2. **Quarantine Affected Systems:** Isolate any systems where the malware is detected to prevent further data exfiltration.
3. **Remove Persistence:** Remove the registry key responsible for launching the keylogger on startup.
4. **Enhance Detection Rules:** Update SIEM systems to detect keystroke logging behaviour and network anomalies involving suspicious HTTP POST requests.

5. **User Awareness:** Inform users about the dangers of keylogger infections and advise caution when opening executable files.

Scenario 3: Ransomware Analysis

A file named document.pdf.exe has been reported in multiple user workstations, resulting in the encryption of critical files. The ransomware also drops a ransom note demanding payment in Bitcoin. Our task is to reverse engineer the ransomware, understand its encryption methodology and extract indicators of compromise (IoCs).

Step 1: Static Analysis

1. **File Properties and Metadata**

- **File Name:** document.pdf.exe
- **File Size:** 200 KB
- **File Type:** PE32 executable (console) Intel 80386, for MS Windows
- **MD5 Hash:** e4a8f1d5c3b9c0a7a2b5d10f8c9b12d8

Actions Taken:

- **Hash Lookup**
  The hash does not return any results in public repositories, which could mean it's a new or custom variant.
- **Metadata Extraction**
  No noteworthy details about the author or compiler, suggesting that it might be compiled without leaving metadata behind.

2. **Strings Analysis**

We use the strings tool to extract readable strings from the binary. Some interesting findings include:

- **Interesting Strings:**
  - Your files have been encrypted.
  - send RM500 in Bitcoin to this address: 1FfmbHfnpaZjKFvyi1okTjJJusN455paPH
  - AES256
  - C:\Documents\encrypted.txt
  - decrypt.exe

These strings suggest that the malware uses AES encryption and contains details about the ransom note and payment method.

3. **PE Structure Analysis**

Using *PEStudio*, we examine the Portable Executable structure:

- **Imported Functions:**
    - CreateFileA
    - ReadFile
    - WriteFile
    - CryptEncrypt
    - CryptAcquireContext

The imports reveal that the malware uses Windows CryptoAPI for encryption. Functions like CreateFileA and WriteFile are associated with file manipulation, confirming that the malware is likely encrypting files.

Step 2: Dynamic Analysis

1. Sandbox Execution

We run the ransomware in a controlled environment using *Cuckoo Sandbox*. Here are the observed behaviours:

- **File Encryption**
  The ransomware begins encrypting files on the user's desktop and in the Documents folder. Files are renamed with a .encrypted extension (e.g., document.docx becomes document.docx.encrypted).
- **Ransom Note Creation**
  After encryption, the ransomware drops a ransom note (READ_ME.txt) in every affected directory. The content includes:
    - **Ransom Message:**

      Your files have been encrypted!

      Send RM 500 in Bitcoin to the following address: 1FfmbHfnpaZjKFvyi1okTjJJusN455paPH

      After payment, email us at decryptme@ransommail.com with your transaction ID to receive the decryption key.
- **No Immediate Network Activity**
  Initially, no network communication is observed, which suggests that this ransomware doesn't immediately contact a C2 server. However, this could happen later (e.g., after payment is made).

2. File System and Registry Monitoring

- **Persistence Mechanism**
  The ransomware adds an entry in the Windows registry to ensure persistence across reboots:
    - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\Ransomware: C:\Users\Victim\document.pdf.exe
- **File Modification**
  It systematically encrypts files across user directories, leaving only the ransom note unencrypted.

3. Network Traffic Analysis

While no immediate communication is detected during sandbox execution, network monitoring will continue to detect any potential contact with a C2 server, such as checking for ransom payment.

Step 3: Code-Level Analysis

1. Unpacking the Ransomware

The ransomware is packed using a common packer. We use the upx -d command to unpack it and reanalyse the strings and imports after unpacking.

2. Disassembly

Using *IDA Pro*, we disassemble the unpacked binary and inspect its critical functions:

- **Encryption Routine**
  The ransomware uses AES-256 for encryption, leveraging the Windows CryptoAPI (CryptEncrypt). It generates a unique AES key for each victim and stores it locally in memory during execution.
- **Ransom Note Creation**
  A function is dedicated to generating the ransom note and placing it in each directory that contains encrypted files.
- **File Handling**
  The ransomware iterates through all user files, encrypting them one by one and renames them with a .encrypted extension.

3. Decryption Key Handling

The malware doesn't store the decryption key on the local system but indicates that after payment, the key would be sent manually via email to the victim. This method further complicates recovery efforts without paying the ransom.

Step 4: Memory Forensics

Using *Volatility*, we analyse a memory dump taken during the malware's execution. Key findings:

- **Extracted AES Key**
  By examining the memory during encryption, we are able to extract the AES encryption key used to encrypt the files.
- **Ransomware Configuration**
  We extract configuration details that reveal the ransomware's Bitcoin wallet address and email contact.

Indicators of Compromise (IoCs)

File Hashes:

- e4a8f1d5c3b9c0a7a2b5d10f8c9b12d8 (ransomware executable)

File System:

- C:\Documents\READ_ME.txt (ransom note)
- Files with .encrypted extension (e.g., document.docx.encrypted)

Registry Keys:

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\Ransomware

Network Indicators:

- Bitcoin Address: 1FfmbHfnpaZjKFvyi1okTjJJusN455paPH
- Email: decryptme@ransommail.com

Behavioural Indicators:

- File encryption and renaming
- Creation of ransom notes in affected directories

Step 5: Final Report

Summary

This ransomware encrypts user files using AES-256 and demands a ransom payment of RM500 in Bitcoin in exchange for the decryption key. The ransomware does not immediately communicate with a command-and-control (C2) server and the payment process relies on manual interaction through email.

Recommendations:

1. **Block Bitcoin Wallet and Email Address:** Report the Bitcoin wallet address 1FfmbHfnpaZjKFvyi1okTjJJusN455paPH and the email address decryptme@ransommail.com to relevant authorities.
2. **Quarantine Affected Systems:** Immediately isolate any infected systems to prevent further encryption.
3. **Restore from Backups:** If possible, restore encrypted files from secure backups instead of paying the ransom.
4. **Patch Vulnerabilities:** Ensure all systems are patched to prevent future exploitation.
5. **Enhance Detection Rules:** Update SIEM systems to detect file renaming patterns and anomalous file encryption activity.

Scenario 4: Banking Trojan Malware

A suspicious file, update.exe, was found on an employee's workstation after they reported unusual behaviour while accessing their online banking account. We need to reverse engineer this malware to understand how it operates, extract Indicators of Compromise (IoCs) and determine its impact.

Step 1: Static Analysis

1. **File Properties and Metadata**

- **File Name:** update.exe
- **File Size:** 180 KB
- **File Type:** PE32 executable (GUI) Intel 80386, for MS Windows
- **MD5 Hash:** f3a1e4b78dc9e2ef28a4d6fcb41b5a23

Actions Taken:

- **Hash Lookup**
  The hash does not return any known results, indicating a new or modified variant.
- **Metadata Extraction** The malware does not contain any notable metadata about its authorship or compilation, which is typical for custom-packed trojans.

2. **Strings Analysis**

We use the strings tool to extract readable text from the binary. The following strings stand out:

- **Interesting Strings:**
  - inject.dll
  - https://bank-login-secure.com
  - GetForegroundWindow
  - sendPOST
  - bank_credentials.txt
  - user32.dll

These strings suggest the malware is injecting code into a web browser (inject.dll) and is likely using sendPOST to send stolen credentials to a malicious server. The GetForegroundWindow function implies that the malware monitors which window is currently active, likely waiting for the browser to access specific banking websites.

3. **PE Structure Analysis**

Using *PEStudio*, we examine the PE structure of the binary:

- **Imported Functions:**
    - GetForegroundWindow
    - WriteFile
    - sendPOST
    - CreateFileA
    - LoadLibraryA
    - GetProcAddress

The imports suggest that the malware is designed to monitor active windows, create and write to files and communicate with a remote server via HTTP POST requests. The presence of LoadLibraryA and GetProcAddress indicates the loading of dynamic link libraries (DLLs) at runtime, which may point to malicious code injection.

Step 2: Dynamic Analysis

1. **Sandbox Execution**

We execute the malware in an isolated sandbox environment using *Cuckoo Sandbox*. The following behaviours are observed:

- **DLL Injection**
  The malware drops and loads a file named inject.dll into the web browser's memory space (e.g., Google Chrome or Firefox).
- **Window Monitoring**
  The malware continuously monitors the active window using GetForegroundWindow and triggers further actions when the browser accesses URLs related to banking sites.
- **Credential Capture**
  Once the victim accesses a banking website, the malware injects JavaScript into the webpage, capturing login credentials (username, password, etc.). These credentials are then saved to a local file, bank_credentials.txt, in the user's AppData folder.
- **Data Exfiltration**
  After saving the credentials locally, the malware initiates a POST request to the malicious server https://bank-login-secure.com/upload to exfiltrate the captured data.

2. **File System and Registry Monitoring**

- **File Drops**
  The malware drops two files:
    - inject.dll (browser injection component)
    - bank_credentials.txt (captured credentials file)
- **Persistence Mechanism**
  The malware ensures persistence by adding itself to the registry under:
    - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\TrojanBanker: C:\Users\Victim\AppData\Roaming\update.exe

Step 3: Code-Level Analysis

1. Disassembly

Using *IDA Pro*, we disassemble the binary and focus on key functions:

- **DLL Injection**
  The function responsible for injecting inject.dll into the web browser's memory space is identified. This DLL hooks browser functions to monitor user input on banking websites.
- **Credential Logging**
  The malware uses a custom JavaScript injection to intercept the login fields on targeted banking websites. The captured credentials are logged in plain text to bank_credentials.txt.

2. JavaScript Injection

We extract and analyse the injected JavaScript from the DLL. It includes the following functionality:

- **Field Monitoring**
  The script monitors the login and password fields on the targeted banking pages.
- **Keylogger Functionality**
  If the fields are filled, the JavaScript logs the entered keystrokes and sends the data to the malicious server.

3. Network Traffic Analysis

Using *Wireshark*, we inspect network traffic during the malware's operation:

- **Outbound Traffic**
  We observe HTTP POST requests to https://bank-login-secure.com/upload, which contain the stolen credentials in the payload. The data is exfiltrated in plain text, making it easy to intercept.

Step 4: Memory Forensics

We use *Volatility* to analyse a memory dump of the infected system:

- **Injected Code**
  We locate the injected inject.dll within the browser's memory space. Extracting and examining the DLL reveals further code responsible for monitoring browser activity and injecting malicious JavaScript.
- **Stored Credentials**
  We confirm that the captured credentials are stored in memory temporarily before being written to bank_credentials.txt and exfiltrated.

Indicators of Compromise (IoCs)

File Hashes:

- f3a1e4b78dc9e2ef28a4d6fcb41b5a23 (malware executable)

File System:

- C:\Users\Victim\AppData\Roaming\inject.dll
- C:\Users\Victim\AppData\Roaming\bank_credentials.txt
- C:\Users\Victim\AppData\Roaming\update.exe

Registry Keys:

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\TrojanBanker

Network Indicators:

- https://bank-login-secure.com/upload
- IP Address: 192.168.1.100 (C2 Server)

Behavioural Indicators:

- DLL injection into the browser process
- Monitoring of active windows for banking websites
- Creation of bank_credentials.txt with captured credentials

Step 5: Final Report

Summary

This malware is a banking trojan that monitors the victim's browser activity and injects malicious code into banking websites. It captures the victim's credentials and

exfiltrates them to a remote server. The malware also establishes persistence through the Windows registry, ensuring it runs on system startup.

Recommendations:

1. **Block Malicious Domains and IPs:** Block all outbound traffic to https://bank-login-secure.com and the associated IP address 192.168.1.100.
2. **Quarantine Affected Systems:** Immediately isolate systems where the malware is detected to prevent further credential theft.
3. **Reset Compromised Credentials:** Advise all affected users to reset their online banking credentials and implement multi-factor authentication where possible.
4. **Remove Persistence Mechanisms:** Delete the registry entry and the malware files from affected systems.
5. **Enhance Browser Security:** Advise users to install security extensions that can detect malicious injections in their browsers.

INCIDENT RESPONSE AND REVERSE ENGINEERING SIMULATIONS

Scenario 1

Your company's SIEM alerts you to suspicious behaviour originating from a user workstation. The user downloaded an email attachment which executed malware, leading to abnormal outbound traffic to a foreign IP. The malware has reportedly been identified as a potential variant of a known Remote Access Trojan (RAT).

Your task is to:

1. Conduct an incident response to contain, analyse and mitigate the threat.
2. Reverse engineer the malware to understand its functionality and extract IoCs for further defence.

Step 1: Incident Response

1. **Detection and Identification**

   - **SIEM Alert Triggered**
     Your QRadar SIEM flagged suspicious traffic originating from UserPC1. The traffic consists of outbound communication to IP 198.51.100.77 over TCP port 443 with a high volume of encrypted traffic (SSL).
   - **Threat Categorisation**
     The activity aligns with a command-and-control (C2) communication pattern. After an initial triage, the team identifies the malware as a variant of a Remote Access Trojan (RAT).

2. **Containment**

   - **Network Containment**
     Immediately isolate UserPC1 from the network to prevent further communication with the external IP.
   - **Host Containment**
     Place the affected workstation into a containment VLAN or physically disconnect it from the network. Disable all user accounts on the machine to avoid further misuse.

3. **Initial Analysis**

   - **Memory Dump Collection**
     You initiate a memory dump using tools like *FTK Imager* to preserve the state of the machine for analysis.

- **Disk Imaging**
  Create a full disk image of the system to capture any changes made by the malware.

4. **Eradication**

- **Malware Removal**
  Use antivirus and malware removal tools like *Malwarebytes* to eradicate any known instances of the malware. Perform registry cleanups and remove persistence mechanisms.

- **Patch Vulnerabilities**
  Identify any software vulnerabilities that the attacker may have exploited. Apply the latest patches to the operating system and any software that could be at risk.

5. **Recovery**

- **System Restoration**
  Restore the affected system from known good backups.

- **Credential Resets**
  Force password changes for any accounts that were accessed on the compromised machine.

- **Monitoring and Hardening**
  Enhance monitoring on all critical systems, especially for indicators related to the detected malware. Implement stronger endpoint protections like EDR solutions.

6. **Lessons Learned**

- **Post-Incident Review**
  Conduct a review with the incident response team to identify areas of improvement in detection, containment or eradication. Share the findings with the relevant teams to enhance defences.

Step 2: Reverse Engineering

Now that the incident has been handled, let's reverse engineer the malware for a deeper understanding of its capabilities.

1. **Static Analysis**

a. Binary File Information

- **File Name:** rundll32.exe (disguised as a legitimate Windows process)
- **File Size:** 120 KB

- **File Type:** PE32 executable (DLL) Intel 80386, for MS Windows
- **MD5 Hash:** d41d8cd98f00b204e9800998ecf8427e

b. Metadata and Structure

- Using *PEStudio*, you inspect the binary structure. No suspicious metadata is found (e.g., compilation details have been stripped). However, the file imports several functions related to network communication and process injection:
    - WSASend
    - WSARecv
    - CreateRemoteThread
    - LoadLibrary

c. Strings Analysis

Using the strings command, you extract readable text from the binary. Some notable strings include:

- C2_Server_IP: 198.51.100.77
- Wininet.dll
- ShellExecuteA
- cmd.exe /c start http://malicious.com

These strings indicate that the malware communicates with a C2 server and uses ShellExecuteA to potentially launch other executables or URLs on the victim machine.

2. Dynamic Analysis

a. Controlled Execution in Sandbox

You execute the malware in a sandbox environment like *Cuckoo* or *Any.Run*. Key observations:

- **Process Injection**
  The malware injects code into explorer.exe, allowing it to run within the context of a trusted process.
- **C2 Communication**
  The RAT initiates outbound connections to the IP 198.51.100.77 using SSL encryption. It opens a reverse shell, allowing the attacker to execute commands on the victim machine.

- **Keylogging and Screenshot Capture**
  You observe the malware capturing keystrokes and taking screenshots at intervals, which are then exfiltrated to the C2 server.

b. Registry and Persistence Monitoring

- **Registry Changes**
  The malware creates a new registry entry to maintain persistence across reboots:
    - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\RAT_Persistence
- **File Drops**
  It drops an additional payload updater.exe in the %AppData% directory to act as a backup execution file in case the primary payload is detected and removed.

c. Network Traffic Analysis

You capture and inspect the malware's network traffic using *Wireshark*:

- **SSL Tunnel**
  The malware establishes a secure SSL tunnel to the C2 server at 198.51.100.77. Although the contents are encrypted, you notice the repetitive POST requests, indicating exfiltration activity.

3. Code-Level Analysis

a. Disassembly in IDA Pro

You disassemble the binary using *IDA Pro*:

- **Encryption Routine**
  The malware uses standard encryption libraries to hide the communication between the victim machine and the C2 server.
- **Remote Code Execution**
  The malware contains routines for executing arbitrary commands received from the C2 server. This explains its remote control functionality and ability to perform lateral movement across the network.

b. Deobfuscation

The binary is packed using a common packer. You unpack it using upx -d, revealing additional functionality like:

- **Password Stealing**
  The unpacked code contains modules designed to steal saved passwords from browsers like Chrome and Firefox.

- **File Exfiltration**
  A module is designed to search for files with specific extensions (e.g., .docx, .xlsx) and exfiltrate them to the C2 server.

4. Memory Forensics

a. Analysing Memory Dump with Volatility

Using *Volatility*, you analyse the memory dump of the infected machine. Key findings:

- **Extracted C2 Commands**
  By inspecting the process memory of explorer.exe, you retrieve snippets of communication between the malware and its C2 server. These include specific commands issued by the attacker, such as file search and screenshot requests.
- **Injected DLL**
  The injected rundll32.exe is still active in memory, reinforcing that the malware remains operational as long as explorer.exe runs.

Indicators of Compromise (IoCs)

File Hashes:

- d41d8cd98f00b204e9800998ecf8427e (RAT binary)

File System:

- C:\Users\Victim\AppData\Roaming\updater.exe
- C:\Windows\System32\rundll32.exe

Registry Keys:

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\RAT_Persistence

Network Indicators:

- C2 Server IP: 198.51.100.77
- SSL Communication on Port 443

Behavioural Indicators:

- Process injection into explorer.exe
- Periodic screenshots and keylogging activities
- Persistent outbound connections to C2 server

Step 3: Final Report

Summary

The incident involved a Remote Access Trojan (RAT) that executed on UserPC1, communicating with an external IP (198.51.100.77) over SSL. The RAT exhibited keylogging, file exfiltration and remote command execution capabilities. Reverse engineering of the binary revealed the use of process injection and password stealing modules.

Recommendations:

1. **Block External IPs:** Immediately block all outbound connections to 198.51.100.77 at the network perimeter.
2. **Monitor Network Traffic:** Implement deep packet inspection for SSL traffic to detect encrypted C2 communications.
3. **Remove Malware:** Use anti-malware tools to identify and remove the RAT and associated files from infected systems.
4. **Patch and Harden Systems:** Ensure that all systems are patched against known vulnerabilities and harden endpoint defences with enhanced EDR tools.

Scenario 2

An energy company detects a ransomware attack that has encrypted files on critical SCADA systems. The attacker demands a ransom of 10 Bitcoin to decrypt the data. The ransomware variant is identified as "DarkBolt," a new strain with advanced evasion techniques. You need to respond to the incident and reverse engineer the ransomware to uncover its encryption methods, persistence mechanisms and potential recovery solutions.

Step 1: Incident Response

1. **Detection and Identification**
    - **SIEM Alert Triggered**
      Splunk flags suspicious file access on a SCADA server (SCADA-SRV1) at 3:45 AM. The server started mass file encryption, targeting .xml, .cfg and .log files.
    - **Threat Categorisation**
      Files are renamed with the .darkbolt extension. The ransom note, READ_ME.txt, is placed in every directory with instructions to pay 10 Bitcoin to a provided wallet address.
    - **Malware Identification**
      The ransom note contains markers suggesting a new ransomware variant, "DarkBolt."
2. **Containment**
    - **Network Containment**
      Immediately isolate SCADA-SRV1 and other potentially compromised systems from the network to prevent further spread of the ransomware.
    - **Backup Protection**
      Ensure that offline backups are secure and isolated from the infected network.
    - **Host Containment**
      Disconnect all critical SCADA systems from external networks and prevent any unnecessary communication with external IPs.
3. **Initial Analysis**
    - **Ransomware Payload**
      Forensic imaging of the infected system is conducted and the darkbolt.exe executable is identified as the likely culprit.
    - **Encryption Status**
      Approximately 70% of the critical configuration files have been encrypted, while some log files remain unaffected.

4. **Eradication**

   - **Terminate Malicious Processes**
     Using *Task Manager* and *Process Explorer*, kill the darkbolt.exe process on the compromised systems.

   - **Quarantine Malware**
     Isolate the malware sample for analysis and remove it from the infected systems using antivirus tools.

5. **Recovery**

   - **System Restoration**
     Restore affected SCADA systems using backups from before the ransomware attack.

   - **Data Recovery Attempt**
     Attempt to recover encrypted files from shadow copies or snapshots, if available.

   - **Password Resets**
     Reset credentials and ensure multi-factor authentication (MFA) is enabled for all critical systems to reduce the risk of lateral movement by the attacker.

6. **Lessons Learned**

   - **Review and Fortify Defences**
     Conduct a detailed review of the incident to identify weaknesses in network security, such as how the ransomware bypassed defences. Ensure additional segmentation of the SCADA network and stronger firewall rules.

   - **Backup and Restore Process**
     Evaluate the current backup process to ensure that regular, offline backups are conducted for critical systems.

Step 2: Reverse Engineering

1. Static Analysis

a. Binary File Information

- **File Name:** darkbolt.exe
- **File Size:** 512 KB
- **File Type:** PE32 executable for MS Windows
- **MD5 Hash:** fc3ff98e8c6a0d3087d515c0473f8677

b. Metadata and Structure

Using *PEStudio* and *ExeinfoPE*, you observe that the ransomware binary is packed with *UPX*. Upon unpacking it, the following imports are noted:

- CryptEncrypt
- VirtualAlloc
- NtCreateFile
- DeleteFileA

c. Strings Analysis

Using strings, you extract several readable text segments from the binary:

- C2_Server_Domain: hxxp://darkbolt-c2.io
- Encrypted_File_Extension: .darkbolt
- Bitcoin_Wallet_Address: 1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2
- Your_Files_Have_Been_Encrypted

These strings confirm that DarkBolt communicates with a C2 server and demands Bitcoin as ransom. The .darkbolt extension is applied to encrypted files.

2. Dynamic Analysis

a. Controlled Execution in Sandbox

You execute the malware in a sandbox environment (e.g., *Any.Run*). Observations include:

- **Encryption Behaviour**
  DarkBolt starts by scanning all logical drives for target file types (.xml, .cfg, .log) and applies AES encryption.
- **Key Generation**
  The malware generates an encryption key on the fly, which is then encrypted using an RSA public key hardcoded in the binary.
- **Ransom Note Creation**
  It generates the ransom note READ_ME.txt and places it in every folder with encrypted files.
- **Network Communication**
  DarkBolt attempts to connect to its C2 server at hxxp://darkbolt-c2.io to send the victim's system ID and encryption key.

b. Network Traffic Analysis

Using *Wireshark*, you capture and inspect the ransomware's network traffic:

- **C2 Communication**
  The malware sends an HTTP POST request containing the encrypted AES key and the victim's system ID. The server responds with a confirmation of successful encryption.
- **No Exfiltration Detected**
  Unlike typical ransomware, DarkBolt doesn't appear to exfiltrate data its primary focus is encryption.

3. Code-Level Analysis

a. Disassembly in IDA Pro

Disassembling the binary in *IDA Pro*, you find:

- **AES Encryption Routine**
  The malware uses the Windows Cryptography API to apply AES-256 encryption to target files.
- **RSA Public Key**
  Embedded within the binary is an RSA public key used to encrypt the AES key. The RSA key pair prevents decryption without the private key, which is presumably held by the attacker.

b. Decryption Logic

- **AES Key Protection**
  The AES encryption key for each file is locally stored but encrypted using the hardcoded RSA public key. Without access to the corresponding RSA private key (held by the attackers), decrypting the files would be impossible through normal means.

4. Memory Forensics

a. Analysing Memory Dump with Volatility

Using *Volatility* on the memory dump, you locate:

- **Encryption Key in Memory**
  The AES encryption keys are present in memory during execution, which offers an opportunity for key recovery if the memory dump is taken while the ransomware is actively running.
- **Injected Code**
  The ransomware injects a monitoring thread into svchost.exe, ensuring that encryption resumes if the malware process is prematurely terminated.

Indicators of Compromise (IoCs)

File Hashes:

- fc3ff98e8c6a0d3087d515c0473f8677 (DarkBolt ransomware binary)

File System:

- Files renamed with .darkbolt extension
- Ransom note: READ_ME.txt in every directory

Network Indicators:

- C2 Server Domain: hxxp://darkbolt-c2.io
- Bitcoin Wallet Address: 1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2

Behavioural Indicators:

- Mass file encryption targeting .xml, .cfg, .log file types
- Network traffic to C2 server domain for key exchange

Step 3: Final Report

Summary

The DarkBolt ransomware attack resulted in file encryption on critical SCADA systems, with the threat actor demanding a ransom of 10 Bitcoin. The ransomware uses AES-256 encryption and RSA public key cryptography to secure the encryption keys, preventing decryption without the attacker's private key. Although some encryption keys were recovered from memory, the majority of files remain encrypted due to the robust RSA encryption.

Recommendations:

1. **Block C2 Communication:** Implement firewall rules to block outgoing communication to darkbolt-c2.io.
2. **Key Recovery Tactics:** Focus on memory forensics for potential key recovery during live infections, capturing active AES keys before they are erased from memory.
3. **Backup and Restore:** Prioritise isolated backups and faster restore processes to mitigate downtime during future incidents.
4. **Segmentation:** Further segment SCADA systems from corporate networks to reduce the attack surface and contain ransomware infections more effectively.

Scenario 3

A leading financial institution detects unusual network activity originating from one of its workstations. The activity includes communication with a known malicious IP address associated with banking trojans. Upon investigation, the workstation is found to have been infected by a variant of the **"TrickLoader"** trojan. The malware's goal is to steal banking credentials and send them to the attacker's Command and Control (C2) server. Your task is to respond to the incident, analyse the malware and reverse engineer it to understand its functionality.

Step 1: Incident Response

1. **Detection and Identification**

   - **SIEM Alert**
     QRadar generates an alert for outbound communication from a workstation (FIN-WS123) to a suspicious IP address (185.65.204.100). The communication pattern is consistent with that of banking trojans.

   - **Suspicious Process**
     A previously unseen process, trickload.exe, is running on the system and was executed from a suspicious folder: C:\Users\Public\Libraries\trickload.exe.

   - **Malware Identification**
     Based on initial indicators, the malware is identified as a variant of the banking trojan family, TrickLoader.

2. **Containment**

   - **Network Containment**
     Immediately isolate FIN-WS123 from the network to prevent further exfiltration of sensitive data.

   - **Enterprise-Wide Scan**
     Initiate a scan across the network to detect any other workstations exhibiting similar behaviour.

   - **Host Containment**
     Disable internet access on the affected machine to prevent further communication with the attacker's C2 server.

3. **Initial Analysis**

   - **Malware Payload**
     Conduct a forensic analysis of the workstation. The executable trickload.exe is isolated as the primary malware payload.

- **Credential Dumping**
  The trojan uses tools like Mimikatz to scrape user credentials from memory, specifically targeting bank-related processes.
- **Persistence Mechanism**
  The malware achieves persistence via registry keys and scheduled tasks.

4. **Eradication**

- **Terminate Malicious Processes**
  Using *Task Manager* and *Sysinternals tools*, terminate the trickload.exe process and related tasks.
- **Registry Cleanup**
- Remove persistence mechanisms by deleting suspicious registry entries and disabling any associated scheduled tasks.
- **Malware Removal**
  Remove the malware from the system using anti-malware tools like *Malwarebytes* and perform a system-wide scan to ensure complete removal.

5. **Recovery**

- **Password Resets**
  Reset all credentials associated with the infected machine, especially those linked to financial accounts.
- **System Restoration**
  Reimage the infected workstation to ensure no remnants of the malware remain.
- **Network Hygiene**
  Conduct a full review of firewall rules, proxy logs and other security appliances to block any further attempts at exfiltration or re-infection.

6. **Lessons Learned**

- **SIEM Rule Tuning**
  Strengthen the SIEM rules and alerts to detect anomalous outbound connections to known C2 servers earlier.
- **Credential Protection**
  Implement stricter credential management policies, such as the use of Privileged Access Workstations (PAWs) for handling sensitive accounts.

Step 2: Reverse Engineering

1. **Static Analysis**

a. Binary File Information

- **File Name:** trickload.exe
- **File Size:** 680 KB
- **File Type:** PE32 executable for MS Windows
- **MD5 Hash:** 9d1b1a3df2b57b7481545ae71918b242

b. Metadata and Structure

Using *PEStudio*, you observe that the file is packed with *UPX*. After unpacking, the following imports are visible:

- WSAStartup
- InternetOpenA
- RegSetValueExA
- CreateProcessA

c. Strings Analysis

Running strings on the unpacked binary reveals:

- http://185.65.204.100/c2
- logindata.bin
- %APPDATA%\TrickLoader\config.json
- CreateScheduledTask
- SetRegKeyPersistence

These strings suggest that TrickLoader communicates with a C2 server, stores stolen credentials in a file called logindata.bin and maintains persistence through registry keys and scheduled tasks.

2. Dynamic Analysis

a. Controlled Execution in Sandbox

Executing the malware in a sandbox environment reveals its behaviour:

- **Credential Stealing**
  TrickLoader monitors active processes for any associated with banking websites or applications (e.g., chrome.exe, firefox.exe). It hooks browser processes to capture credentials entered on online banking portals.

- **C2 Communication**
  The malware initiates periodic connections to http://185.65.204.100/c2 to exfiltrate the stolen credentials.

- **Persistence Mechanism**
  It creates a scheduled task named WindowsUpdater that runs the malware on startup. Additionally, it sets a registry key under HKCU\Software\Microsoft\Windows\CurrentVersion\Run.

b. Network Traffic Analysis

Using *Wireshark*, you capture and analyse TrickLoader's network traffic:

- **C2 Communication**
  TrickLoader sends HTTP POST requests containing encrypted data to the C2 server. The data includes banking credentials stored in the logindata.bin file.

- **No Data Exfiltration Yet**
  At the time of capture, no sensitive banking data has been exfiltrated, but credentials were collected and ready to be sent in future transmissions.

3. Code-Level Analysis

a. Disassembly in IDA Pro

Using *IDA Pro* to disassemble the unpacked binary, you uncover:

- **Process Injection**
  TrickLoader injects code into running browser processes using the Windows API function CreateRemoteThread. This allows the malware to capture keystrokes and form data.

- **Encryption Algorithm**
  The stolen data is encrypted using a custom XOR cipher before being transmitted to the C2 server. The encryption key is hardcoded as 0x55AA in the binary.

- **C2 Protocol**
  TrickLoader uses a simple custom protocol for C2 communication, embedding the stolen data in the body of HTTP POST requests.

b. Hooked Functions

- **API Hooking**
  The malware hooks several key API calls in browser processes, such as GetMessageA and SendMessageA, to intercept user input from online banking portals.

4. Memory Forensics

a. Analysing Memory Dump with Volatility

Using *Volatility* to analyse a memory dump of the infected system, you find:

- **Injected Code in Browser Processes**
  TrickLoader injected code into chrome.exe, allowing it to scrape browser history and intercept form submissions.
- **Credential Recovery**
  The decryption key (0x55AA) found in the code allows you to decrypt the stolen credentials stored in memory.

Indicators of Compromise (IoCs)

File Hashes:

- 9d1b1a3df2b57b7481545ae71918b242 (TrickLoader trojan binary)

File System:

- Stolen credentials stored in C:\Users\Public\logindata.bin
- Configuration file: %APPDATA%\TrickLoader\config.json
- Registry Key:
  HKCU\Software\Microsoft\Windows\CurrentVersion\Run\WindowsUpdater

Network Indicators:

- C2 Server IP: 185.65.204.100
- C2 URL: http://185.65.204.100/c2

Behavioural Indicators:

- Scheduled task: WindowsUpdater
- Outbound HTTP POST requests to the C2 server containing encrypted stolen credentials.

Step 3: Final Report

Summary

The TrickLoader trojan was discovered on a workstation within the financial institution's network. The malware's primary objective was to steal banking credentials by hooking browser processes and sending the data to a remote C2 server. Although the malware was detected before sensitive data could be exfiltrated, its persistence mechanisms and credential-stealing capabilities pose a significant risk.

Recommendations:

1. **Block C2 Communication:** Immediately block outbound traffic to 185.65.204.100 at the firewall and proxy levels.
2. **Enhance Credential Security:** Implement credential protection mechanisms like Credential Guard and restrict the use of administrative credentials to reduce exposure.
3. **User Training:** Increase user awareness of phishing attacks and how they can lead to trojan infections.
4. **Advanced Endpoint Detection:** Deploy advanced endpoint protection solutions that can detect and prevent API hooking and process injection.