W24D4 FINAL PROJECT ON MALWARE ANALYSIS

Malware Analysis

Il Malware da analizzare è nella cartella Build_Week_Unit_3 presente sul desktop della macchina virtuale dedicata.

Analisi statica

Con riferimento al file eseguibile Malware_Build_Week_U3, rispondere ai seguenti quesiti utilizzando i tool e le tecniche apprese nelle lezioni teoriche:

- Quanti parametri sono passati alla funzione Main()?
- Quante variabili sono dichiarate all'interno della funzione Main()?
- Quali sezioni sono presenti all'interno del file eseguibile? Descrivete brevemente almeno 2 di quelle identificate
- Quali librerie importa il Malware? Per ognuna delle librerie importate, fate delle ipotesi sulla base della sola analisi statica delle funzionalità che il Malware potrebbe implementare. Utilizzate le funzioni che sono richiamate all'interno delle librerie per supportare le vostre ipotesi.

Malware Analysis

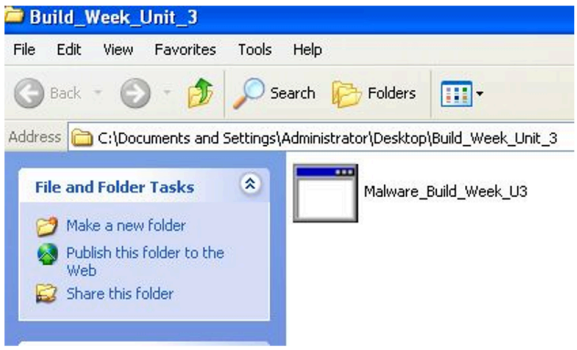
Con riferimento al Malware in analisi, spiegare:

- Lo scopo della funzione chiamata alla locazione di memoria **00401021**
- Come vengono passati i parametri alla funzione alla locazione **00401021**;
- Che oggetto rappresenta il parametro alla locazione **00401017**
- Il significato delle istruzioni comprese tra gli indirizzi **00401027** e **00401029**.
- Con riferimento all'ultimo quesito, tradurre il codice Assembly nel corrispondente costrutto C.
- Valutate ora la chiamata alla locazione **00401047**, qual è il valore del parametro «ValueName»?

Malware Analysis

Analisi dinamica

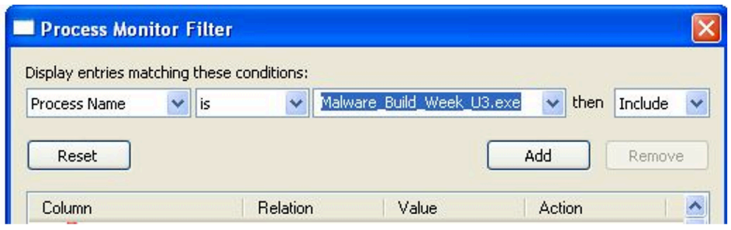
Preparate l'ambiente ed i tool per l'esecuzione del Malware (suggerimento: avviate principalmente Process Monitor ed assicurate di eliminare ogni filtro cliccando sul tasto «reset» quando richiesto in fase di avvio). Eseguite il Malware, facendo doppio click sull'icona dell'eseguibile



Malware Analysis

- Cosa notate all'interno della cartella dove è situato l'eseguibile del Malware? Spiegate cosa è avvenuto, unendo le evidenze che avete raccolto finora per rispondere alla domanda

Analizzate ora i risultati di Process Monitor (consiglio: utilizzate il filtro come in figura sotto per estrarre solo le modifiche apportate al sistema da parte del Malware). Fate click su «ADD» poi su «Apply» come abbiamo visto nella lezione teorica.



Malware Analysis

Filtrate includendo solamente l'attività sul registro di Windows.

- Quale chiave di registro viene creata?
- Quale valore viene associato alla chiave di registro creata?

Passate ora alla visualizzazione dell'attività sul file system.

- Quale chiamata di sistema ha modificato il contenuto della cartella dove è presente l'eseguibile del Malware?

Unite tutte le informazioni raccolte fin qui sia dall'analisi statica che dall'analisi dinamica per delineare il funzionamento del Malware.

1. Parameters Passed to the Main() Function

- Typically, in C/C++ programs, the `main()` function can take two parameters: `argc` (argument count) and `argv[]` (argument vector, an array of strings representing the command-line arguments).
- Without the actual source code, the exact number of parameters is unclear, but if we assume a standard `main()` function, it would likely be two parameters.

2. Variables Declared Inside the Main() Function

- This depends on the specific implementation of the `main()` function. We would need to examine the function's code or disassembly to determine the exact number of variables.
- In assembly, variables might correspond to specific memory locations or registers being used.

3. Executable File Section

- Common sections in an executable include:
  - `.text`: Contains the executable code or instructions that the CPU executes.

        - `.data`: Stores global and static variables initialized by the programmer. This section contains data that can be modified.
        - `.rdata`: This section typically contains read-only data, such as string literals or constants used by the program.
        - `.rsrc`: Contains resources like icons, images, and menus that the executable might use.

4. Imported Libraries

    - For the malware in question, the imported libraries might include:
        - `kernel32.dll`: Often used by malware for basic system operations, such as managing memory, processes, and threads.
        - `advapi32.dll`: Used for advanced Windows API functions, particularly around security and registry operations. This could allow the malware to manipulate system security settings and achieve persistence.
- Hypotheses Based on Imported Libraries:
    - `kernel32.dll`: The malware might use functions like `CreateFile`, `ReadFile`, `WriteFile`, and `VirtualAlloc`to manipulate files and memory.
    - `advapi32.dll`: Functions like `RegOpenKeyEx`, `RegSetValueEx`, and `AdjustTokenPrivileges` suggest the malware is interacting with the Windows Registry, possibly to ensure it runs on startup or to modify security privileges.

5. Memory Location Analysis

- Purpose of the Function Called at 00401021
    - Without seeing the actual assembly code, it’s hard to determine the exact purpose. However, if we are

referencing a standard function call in malware, this could be where the malware is performing a critical operation, such as writing to the registry or allocating memory.

- Parameter Passing to the Function at 00401021
  - In assembly, parameters are typically passed via registers or the stack. For example, the `push` instruction is often used to place parameters on the stack before a function call.
- Object Represented by the Parameter at 00401017
  - This could be a handle to a system object, such as a file, registry key, or a network socket, depending on the context of the program.
- Instructions Between 00401027 and 00401029
  - This likely involves a small set of operations, such as comparing values or branching based on a condition. These instructions could be setting up conditions for the function's operations.
- Assembly to C Translation
  - This involves converting assembly instructions back into a high-level C-like syntax. This can be complex and typically requires a detailed look at the specific instructions.
- Value of the Parameter `ValueName` at 00401047
  - This would be the name of the registry value being set or queried. It could be a string like `"Startup"` or `"Run"` that indicates the registry key used to achieve persistence.

6. Dynamic Analysis

- Setting Up the Environment
  - Before executing the malware, tools like Process Monitor should be reset to clear any filters. This allows capturing all system activity when the malware is run.

    - The malware is executed by double-clicking its icon, which will trigger its payload.
- Observations in the Malware's Directory
    - After execution, you might notice new files created or existing files modified. These changes could include the creation of new executable files, scripts, or logs.
- Process Monitor Analysis
    - Registry Key Created: You would need to filter for registry activities. Common registry keys modified by malware include those under `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`.
    - Value Associated with the Key: This would typically be the path to the malware executable or a script that ensures the malware runs on startup.
- File System Activity
    - The system call that modified the directory might be `CreateFile`, `WriteFile`, or `MoveFile`, which the malware uses to write new files or change existing ones.

Combining Static and Dynamic Analysis

The combined analysis from static (e.g., section headers, imported functions) and dynamic analysis (e.g., system calls, registry modifications) gives a comprehensive understanding of how the malware operates:

- Persistence: Through registry modifications to ensure the malware runs on startup.
- File Manipulation: The malware might create or modify files in its directory, possibly to deploy additional payloads or to hide its presence.
- System Impact: By analyzing these behaviors, you can infer how the malware spreads, maintains persistence, and what its potential goals are (e.g., data theft, system compromise).