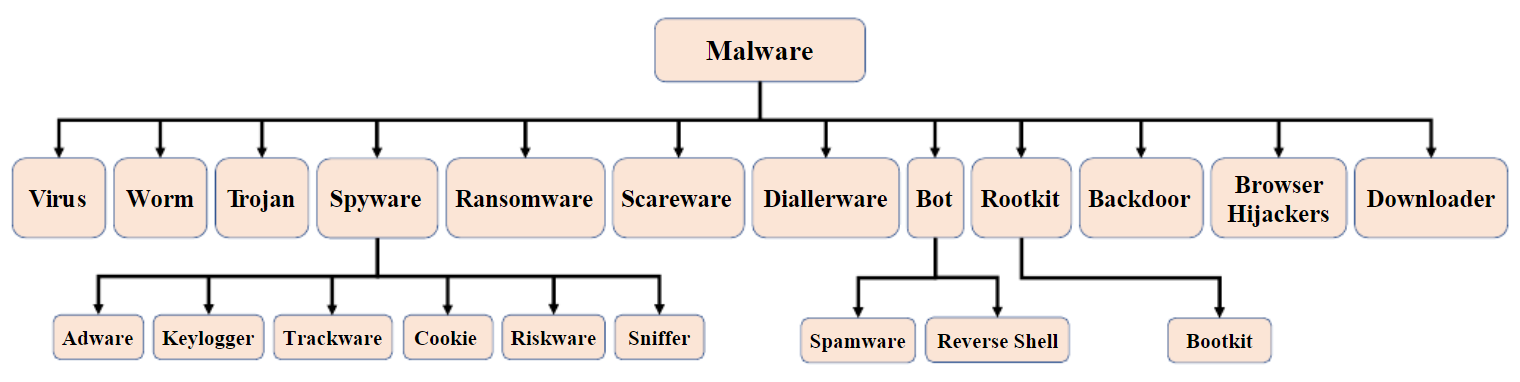
What Is malware?

It is a software or binary that is malicious in nature, designed to damage a computer system without user’s informed consent. And used by the attackers to perform a variety of malicious action like spying on the target, data exfiltration end data encryption and destructions.

Generally, a malware is categorized into following categories:



Malware analysis:

Malware analysis is the process of determining the purpose and characteristics of a given malware sample such as virus, worm or Trojan horses. The tools used for malware analysis can basically be categories.

* Static
* Dynamic
* Code Analysis - the process of analyzing/reverse engineering assembly code and this can be both statically and dynamically done.
* Behavioral Analysis - in this process of analyzing and monitoring the malware after execution. IT involves the monitoring the process registry entry and network monitoring to determine the working of malware.

Static malware analysis:

the analyzing of software without executing it is termed as static analysis and static analysis technique can be applied on different representation of a program with an objective to extract as much metadata from malware as possible there are different techniques used for static malware analysis such as:

* File fingerprinting- this can be done with the help of cryptographic hash operation such as MD5, SHA 01, end SHA 256. And mostly used in digital forensic.
* file format - by extracting the metadata of a given file format the useful information can be gathered easily, in Windows operating system the extraction of the file information in the form of window binary which is typically in PE format (portable executable) a lot of information can be extracted easily.
* antivirus scanning - the AV scanning can be done, there is a likely to be a malware in a binary code file. The AV scanner is a time consuming, but it becomes necessity sometimes.
* Packer detection - nowadays the malware is mostly distributed in obfuscated form after taking a program look much different forms then is static analysis perspective and in order to extract the other metadata it will become hard to recover it so there is certain unpacked such as PEiD2, there is accordingly no generic unpacker present this makes a major challenge of a static malware analysis.
* Disassembling - a major part of a static analysis can be done using a disassembling of a given binary by using a tool such as IDA pro. Which will reverse the machine code to the assembly language.

The main advantage of static malware analysis is it will cover all the possible execution part of the malware sample and is generally safer than a dynamic analysis as the source code is not actually executed but now a days the malware is Present in officiated format which reduce the static analysis technique in order to find malwares.

Dynamic malware analysis

A given malware sample can be executed within a control environment and monitoring its action in order to analyze the malicious behavior which is called dynamic malware analysis. Since dynamic malware analysis is performed during the runtime and malware unpack itself. However, the main drawback is called dormant code where it can’t be able to analyze the behavior of the program because dormant code is not executed. There are two basic approaches for dynamic malware analysis characterized by:

* Analyzing the difference between defined points - A given malware sample is executed for certain create of time and after completion off them code with a modification the system is going to analyzed by the comparison with the initial system state.
* observing the runtime behavior - In this approach malicious activity launched by the malicious application are monitored during runtime by a specialized tool.

Malware Analysis Tool:

* Process Monitor – This tool is a part of the Microsoft TechNet website (open-source software). The tool monitors and display in real-time all file system activity on a Microsoft Windows OS. Used to detect failed attempts to read and write registry keys and use to filtering on specific keys, processes, process IDs, and values. With the help of filemon it become is rather noisy and picks up hundreds of file changes by an idle window system. And become too handy in order to run and executing of binary.
* Norman Sandbox -The Norman Sandbox is a dynamic malware analysis solution which executes the sample in a tightly controlled virtual environment that simulates a Windows operating system.

Malware detection:

* Signature- based detection- An early staple of antivirus programs was signature detection where a unique code pattern or hash of a known malicious file is known and recorded. Once this signature is discovered again, the file containing it can be flagged by the antivirus.

As malware became more sophisticated, malware authors began using new techniques, like polymorphism, to change their pattern each time their creation spread from one system to the next. As such, this minimized the effectiveness of a simple signature detection. Researchers then supplemented this with heuristic detection that judges the code based on its behavior. When anything starts acting out of the ordinary, it sets off alarm bells.

* **Cloud-based detection** – Cloud based detections shift the identification work from the individual device to the cloud. This frees up computer space for more productive tasks and enables security firms to keep their detection methodologies more hidden from the cyber-criminals. By adding AI-enhanced machine learning into the mix, security firms can sort and sift through potential malware much faster and more in-depth than in the past, saving their manual ID work for new and emerging threats.

Application Log:

Code available: C:\Users\Z004SU2S\Desktop\Adversary-in-the-Middle

*This code will first import the necessary libraries, including the logging library for logging events, the os library for accessing the file system, and the re library for regular expressions.*

*Next, the code defines a function called monitor\_application\_logs() that takes a path to an application log file as input and returns a list of events that may indicate an AiTM attack. The function uses a regular expression to search the log file for lines that indicate that a setting has been changed for a network protocol or service.*

*Finally, the code defines a main function that calls the monitor\_application\_logs() function and prints the list of events to the console.*

**Line 1**: This line imports the **logging** library, which is used for logging events.

**Line 2**: This line imports the **os** library, which is used for accessing the file system.

**Line 3**: This line imports the **re** library, which is used for regular expressions.

Line 6: This line defines a function called **monitor\_application\_logs()** that takes a path to an application log file as input and returns a list of events that may indicate an AiTM attack. The function uses a regular expression to search the log file for lines that indicate that a setting has been changed for a network protocol or service.

Line 7: This line defines the argument **log\_file** for **the monitor\_application\_logs()** function. The **log\_file** argument is the path to the application log file.

Line 8: This line defines a list called **events**. The **events** list will store the events that are found in the log file.

Line 9: This line opens the log file in read mode.

Line 10: This line iterates over the lines in the log file.

Line 11: This line uses a regular expression to search the current line for a match. The regular expression matches lines that contain the text **changed settings for** followed by two other words.

Line 12: This line checks if the regular expression found a match. If it did, the match object is stored in the variable match.

Line 13: This line appends the match object to the **events** list.

Line 14: This line closes the log file.

Line 15: This line returns the **events** list.

Line 17: This line defines the main function. The main function calls the **monitor\_application\_logs()** function and prints the list of events to the console.

Line 18: This line defines the **log\_file** variable. The **log\_file** variable is the path to the application log file.

Line 19: This line calls the **monitor\_application\_logs()** function and stores the results in the **events** variable.

Line 20: This line iterates over the events in the **events** list.

Line 21: This line logs the event to the console.

Line 22: This line is the **if \_\_name\_\_ == "\_\_main\_\_":** statement. This statement ensures that the main function is only executed when the code is run as a script.

This code will first create a regular expression to match known AiTM network management protocols. The code will then monitor for network traffic that matches the pattern. If any traffic is detected that matches the pattern, an alert will be logged.

To run this code, you will need to install the following libraries:

* re - This library provides regular expression support.
* pcapy - This library provides packet capture support.

Network Traffic Content

This code will first create a regular expression to match known AiTM network management protocols. The code will then monitor for network traffic that matches the pattern. If any traffic is detected that matches the pattern, an alert will be logged.

To run this code, you will need to install the following libraries:

* re - This library provides regular expression support.
* pcapy - This library provides packet capture support.