AV Evasion Concepts

A modern antivirus is typically designed around the following components:

- File Engine
- Memory Engine
- Network Engine
- Disassembler
- Emulator/Sandbox
- Browser Plugin
- Machine Learning Engine

Each of these engines work simultaneously with the signature database to rank specific events as either benign, malicious, or unknown.

- The ***file engine*** is responsible for both scheduled and real-time file scans. When the engine performs a scheduled scan, it simply parses the entire file system and sends each file's metadata or data to the signature engine. On the contrary, real-time scans involve detecting and possibly reacting to any new file action, such as downloading new malware from a website. In order to detect such operations, the real-time scanners need to identify events at the kernel level via a specially crafted *mini-filter driver*. This is the reason why a modern AV needs to operate both in kernel and user land, in order to validate the entire operating system scope.
- The ***memory engine*** inspects each process's memory space at runtime for well-known binary signatures or suspicious API calls that might result in memory injection attacks.
- The ***network engine*** inspects the incoming and outgoing network traffic on the local network interface. Once a signature is matched, a network engine might attempt to block the malware from communicating with its *Command and Control* (C2) server.
- To further hinder detection, malware often employs encryption and decryption through custom routines in order to conceal its true nature. AVs counterattack this strategy by *disassembling* the malware packers or ciphers and loading the malware into a **sandbox**, or ***emulator***.
- The ***disassembler*** engine is responsible for translating machine code into assembly language, reconstructing the original program code section, and identifying any encoding/decoding routine. A *sandbox* is a special isolated environment in the AV software where malware can be safely loaded and executed without causing potential havoc to the system. Once the malware is unpacked/decoded and running in the emulator, it can be thoroughly analyzed against any known signature.

- As browsers are protected by the sandbox, modern AVs often employ browser plugins to get better visibility and detect malicious content that might be executed inside the browser.
- Additionally, the machine learning component is becoming a vital part of current AVs as it enables detection of unknown threats by relying on cloud-enhanced computing resources and algorithms.

Bypassing AV Detections

There are mainly two types of evasion techniques - On Disk Evasion and In Memory Evasion.

Let's see On-Disk Evasion first.

On-Disk Evasion: On-disk evasion techniques aim to modify the malware payload on disk to evade signature-based detection by AV engines. Some common methods include:

- **Obfuscation:** Encrypting, packing, or polymorphically modifying the malware code to change its signature and avoid detection. Using obfuscators, cryptors, and protectors to alter the on-disk binary structure.
- **Exploiting Vulnerabilities:** Leveraging vulnerabilities in the AV software itself to disable or bypass protection mechanisms.
- **Environmental Keying:** Detecting the presence of AV or analysis environments (e.g., sandboxes, VMs) and altering the malware's behavior accordingly to evade dynamic analysis.

Now lets see In-Memory evasion:

In-Memory Evasion: In-memory evasion techniques aim to execute malicious code directly in memory without writing it to disk, evading file-based AV scanning. Some common methods include:

- **Process Memory Injection:** Injecting malicious code into legitimate running processes to execute and evade detection.
- **DLL Injection:** Injecting a malicious DLL into a process's memory and executing it without writing it to disk.
- **Reflective Loading:** Loading and executing malicious code directly from memory without relying on traditional Windows loader mechanisms.
