How Stealth Loading Malware Works

Reverse Engineering
Joshua Reynolds
Spring 2023

Loader

An innocent-looking shell program used to start actual malware

Loader

An innocent-looking shell program used to start actual malware. Runs:

- On startup
- cron job (Linux) / Task Scheduler (Windows)
- Plug-and-play drivers
- Autorun on storage media
- ActiveScriptEventConsumer

Starting the Malware

Loader may load the malware from Internet, filesystem, registry, or its own resources

Goal: Start malware without the user or antivirus noticing.

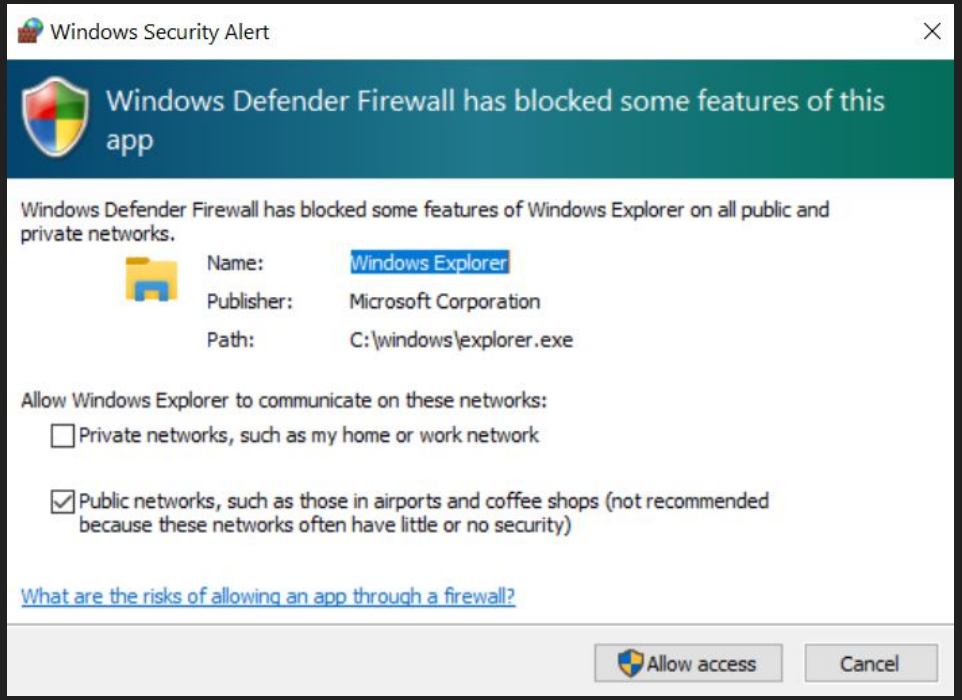
Strategy 1: DLL Injection / Shared Object Injection

Libraries loaded at runtime

- Replace legitimate library with evil twin
- Run a simple program which loads a malicious DLL/SO
- Force another program to load a malicious DLL/SO

Why inside another process?

Why inside another process?



Attacker gains access privileges of that program (Internet, Peripherals, Filesystem, User Privileges)

(SetUID Bit Example)

How malware forces other processes to load DLLs

1. Find the PID of the target process
   a. CreateToolhelp32Snapshot, Process32First, Process32Next
2. OpenProcess (Get process handle)
3. VirtualAllocEx (Allocate extra memory)
4. WriteProcessMemory (Write DLL name)
5. CreateRemoteThread (Start a new thread)
6. DLLMain is automatically run upon load

What will a malware loader doing DLL injection look like?

```
maliciousDLLName = “evil.DLL”;

hVictimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, victimPID);

pBadDLLName = VirtualAllocEx(hVictimProcess,...,strlen(maliciousDLLName))

WriteProcessMemory(VictimProcessHandle,...,maliciousDLLName);

GetModuleHandle(“Kernel32.dll”);

pLoadLibrary = GetProcAddress(...,”LoadLibraryA”);

CreateRemoteThread(hVictimProcess, pLoadLibrary, pBadDLLName);
```

What to do when you find this DLL injection pattern?

What do we want to know?

What to do when you find this DLL injection pattern?

What do we want to know?

1. What is the victim process?
2. What is the malicious DLL?
3. What is in the malicious DLL's DLLMain function?

Strategy 2: Direct Injection

Why only write a DLL name into a victim program's memory space? You could also write code. What code?

Defense: Data Execution Prevention (DEP)

Strategy 2: Direct Injection

Why only write a DLL name into a victim program's memory space? You could also write code. What code?

Probably Shellcode

Defense: Data Execution Prevention (DEP)

Shellcode

A portal to easily execute arbitrary code (scripts are easiest)

- cmd.exe
- mshta.exe
- powershell.exe

Callback Shell

An Internet portal to easily send commands to some shellcode running on a remote computer.

Usually authenticated (malware authors will steal each others' victims!)

Can be disguised as DNS traffic, a webserver, or any other "normal" network function

Strategy 3: Process Replacement

What if the victim process crashes or ends normally? Sometimes malware prefers to take over execution of the entire process.

svchost.exe is a common victim for this

Malware Steps for Process Replacement

1. Start the process in suspended mode
2. Free all the memory in the code section
3. Re-Allocate the memory in the code section
4. Allow the process to begin

Defense: DEP

Strategy 4: Hook Injection

Windows allows hooks to run whenever certain events fire. Malware can inject an asynchronous hook into a victim process.

Types of Hooks

Local Hooks:

- Called whenever this process receives a message

High-Level Remote Hooks:

- Called after this process sends a message (must be in DLL)

Low-Level Remote Hooks:

- Called before this process passes a message to the OS

SetWindowsHookEx

Must specify: type of hook, function pointer, DLL handle, threadID to do the work (0 to select all)

Example: Keylogger

A keylogger will use the events WH_KEYBOARD or WH_KEYBOARD_LL (low level version) and include a small function to write the keystroke to a log

Thread Targeting: Where to put these hooks?

1. If you are a keylogger, choose a remote hook and set TheadID to 0
2. Else if want to be stealthy, choose a threadID belonging to some innocent process.
3. Choose the high-level hook that allows you to load your function from a custom DLL

Strategy 5: The “Detours” Feature

A “feature” that allows you to save in-memory modifications to a PE to be applied every time you run the program.

Strategy 5: The “Detours” Feature

A “feature” that allows you to save in-memory modifications to a PE to be applied every time you run the program. Too good to be true?

Strategy 5: The “Detours” Feature

A “feature” that allows you to save in-memory modifications to a PE to be applied every time you run the program or load a DLL. Too good to be true?

Windows’ competitive advantage is how many things “just work”. Eliminating detours could cause backwards compatibility problems.

Strategy 6: Asynchronous Procedure Call queue injection

Windows is built around GUIs. GUIs are asynchronous. Their threads have to pull jobs off of a queue. Why not add malware code to that queue?

User-Mode APC

Works a lot like CreateRemoteThread()

QueueUserAPC(“evil.dll”, threadID, &LoadLibraryA)

Translation: Next time this threadID is available, please execute LoadLibraryA with the parameter “evil.dll”

Kernel-Mode APC

Sometimes a malicious driver in the OS needs to execute something in user-space. Rootkits can use their kernel-mode privileges to force any thread to accept APCs

Example: the pop-up GUI showing a ransomware demand

Why still vulnerable?

https://xkcd.com/1172/

Modern Windows Defenses

https://learn.microsoft.com/en-us/windows/security/threat-protection/overview-of-threat-mitigations-in-windows-10

Based on Chapter 12 of Sikorski and Honig's *Practical Malware Analysis*