Process Injection Techniques used by Malware

medium.com/csg-govtech/process-injection-techniques-used-by-malware-1a34c078612c

Introduction

Process injection is a camouflage technique used by malware. From the Task Manager, users are unable to differentiate an injected process from a legitimate one as the two are identical except for the malicious content in the former. Besides being difficult to detect, malware using process injection can bypass host-based firewalls and specific security safeguards.

What is Process Injection Used For?

There are various legitimate uses for process injection. For instance, debuggers can use it to hook into applications and allow developers to troubleshoot their programs. Antivirus services inject themselves into browsers to investigate the browser's behaviour and inspect internet traffic and website content.

Can Process Injections Be Used For Malicious Purposes?

Process injections are techniques; they can be used for both legitimate and malicious purposes. Because process injections are well-suited to hiding the true nature of action, they are often used by malicious actors to hide the existence of their malware from the victim. Some of the malicious activities that such actors can hide using process injections include data exfiltration and keylogging. Often, victims fail to realise that malicious files have been uploaded simply because the malicious processes are masked to look like innocuous ones.

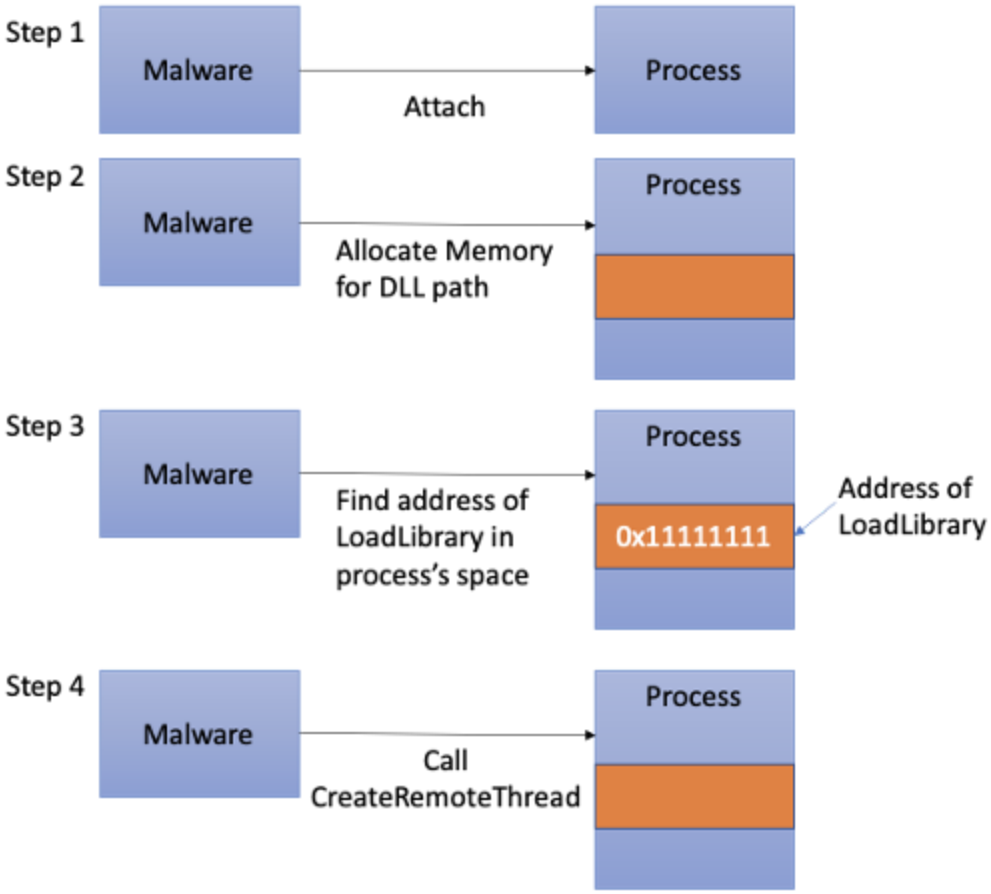
Process Injection Techniques

While process injection can happen on all three major operating systems — Windows, Linux and MacOS — this article will be focussing on Windows.

Technique #1: DLL Injection

A Dynamic Link Library (DLL) file is a file containing a library of functions and data. It facilitates code reuse as many programs can simply load a DLL and invoke its functions to do common tasks.

DLL injection is one of the simplest techniques, and as such, is also one of the most common. Before the injection process, the malware would need to have a copy of the malicious DLL already stored in the victim's system.



Step 1: The malware issues a standard Windows API call (OpenProcess) to attach to the victim process. Due to the privilege model in Windows, the malware can only attach to a process that is of equal or lower privilege than itself.

Step 2: A small section of memory is allocated within the victim process using VirtualAllocEx. This memory is allocated using "write" access. The malware will then issue WriteProcessMemory to store the path of the DLL to that memory location.

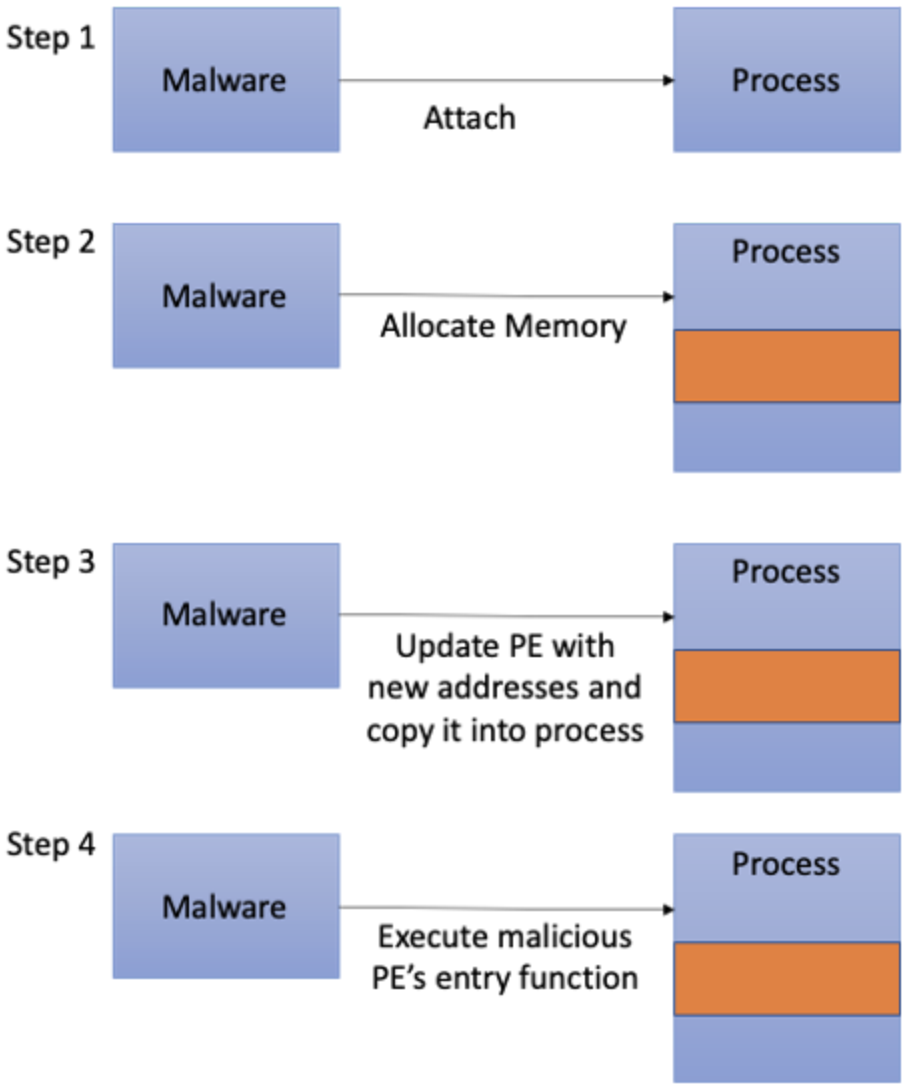
Step 3: The malware looks for the address of the LoadLibrary function within the victim process' space. This address will be used in Step 4.

Step 4: The malware calls CreateRemoteThread, passing in the address of LoadLibrary found in Step 3. It will also pass in the DLL path that it created in Step 2. CreateRemoteThread will now execute in the victim process and invoke LoadLibrary, which in turn loads the malicious DLL. When the malicious DLL loads, the DLL entry method, DLLMain, will be invoked. This will be where malicious activities will take place.

Technique #2: PE Injection

A Portable Execution (PE) is a Windows file format for executable code. It is a data structure containing all the information required so that Windows knows how to execute it.

PE injection is a technique in which malware injects a malicious PE image into an already running process. An advantage of this technique over DLL injection is that this is a disk-less operation, i.e. the malware does not need to write its payload onto disk prior to the injection.



Step 1: The malware gets the victim process' base address and size.

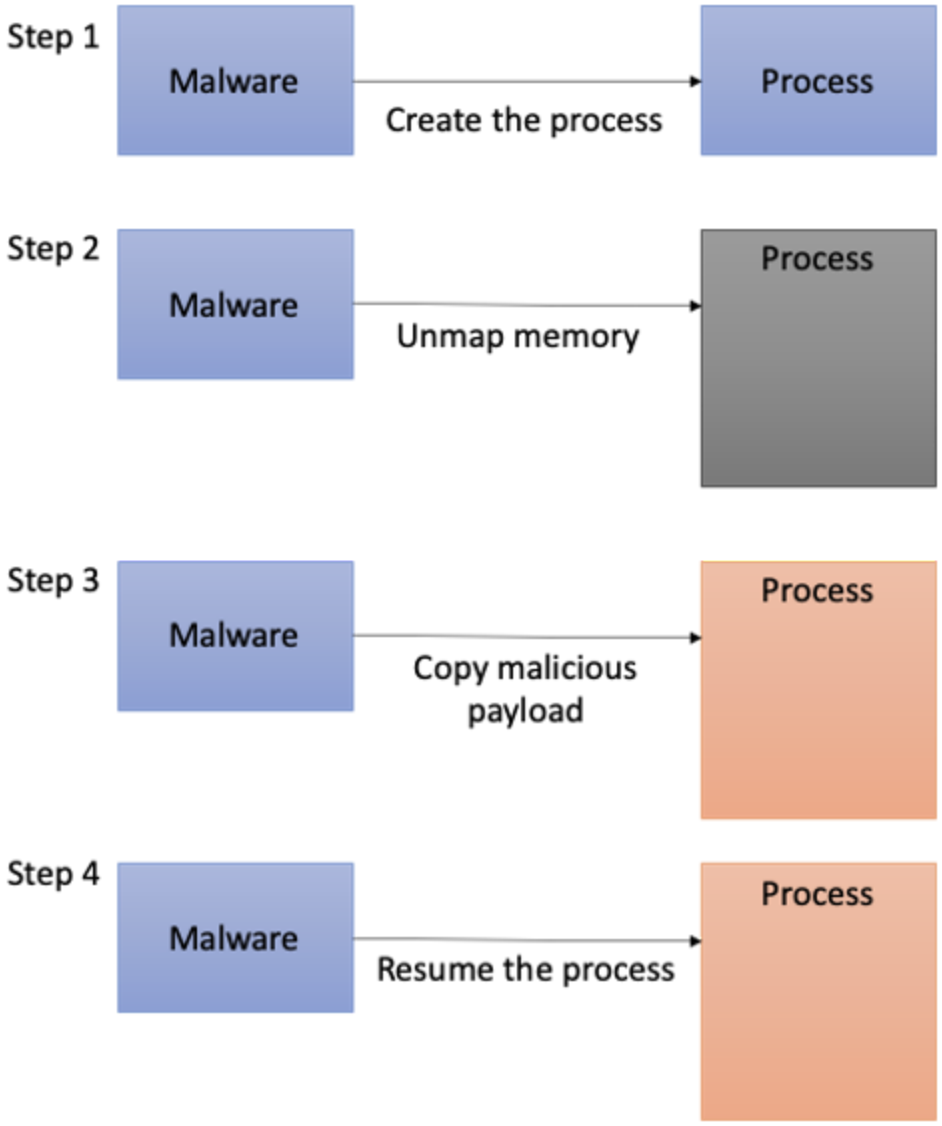
Step 2: The malware allocates enough memory in the victim process to insert its malicious PE image.

Step 3: As the inserted image will have a different base address once it is injected into the affected process, the malware will need to find the victim process's relocation table offset first. With this offset, the malware will modify the image so that any absolute addresses in the image will point to the right functions. Once the malicious PE image has been updated, the malware copies it into the process.

Step 4: The malware looks for the entry function to be executed and runs it using CreateRemoteThread.

Technique #3: Process Hollowing

Unlike the first two techniques, where malware injects into a running process, process hollowing is a technique where the malware launches a legitimate process but replaces the process' code with malicious code. The advantage of this technique is that the malware becomes independent of what is currently running on the victim's system. Furthermore, by launching a legitimate process (e.g. Notepad or svchost.exe), users will not be alarmed even if they were to look through the process list.



Step 1: The malware creates a legitimate process, like Notepad, but instructs Windows to create it as a suspended process. This means that the new process will not start executing.

Step 2: The malware hollows out the process by unmapping memory regions associated with it.

Step 3: The malware allocates memory for its own malicious code and copies it into the process' memory space. It then calls SetThreadContext on the victim process, which changes the execution context of the process to that of the malicious one that was just created.

Step 4: The malware resumes the process; thereby executing the malicious code.

Technique #4: Injection and Persistence via Registry Modification

The Windows Registry is a hierarchical database that stores information required by Windows and programs in order to run properly. The registry stores information such as customisation settings, driver data and startup programs.

The two keys, Appinit_Dlls and AppCertDlls, that malware use for both injection and persistence can be found here:

HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\Appinit_Dlls
HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows\Appinit_Dlls
HKLM\System\CurrentControlSet\Control\Session Manager\AppCertDlls

While managing to add their entries in the registry has far reaching effects, modifying the values of these keys requires the malware to have administrative rights.

Appinit_DLL

The Appinit_DLL registry key allows custom DLLs to be loaded into the address space of every application. This allows software developers an easy way to hook onto system APIs defined in user32.dll that will be used across every application. User32.dll is a system DLL that many graphical applications will import as it contains functions such as controlling dialog boxes or reacting mouse events.

Malware that successfully registers their malicious DLLs in this key will be able to intercept system API calls for every graphical application for nefarious purposes.

To mitigate abuse, Windows 8 and later versions with secure boot enabled have automatically disabled this mechanism. Microsoft does not allow developers to attain certification for applications that rely on this in a bid to discourage developers from abusing this key.

AppCertDlls

This is similar to Appinit_DLL; malware that manages to add their DLLs to this registry key will get to be imported by any application which calls functions like CreateProcess, CreateProcessAsUser, CreateProcessWithLogonW, CreateProcessWithTokenW, and WinExec.

Technique #5: Injection using Shims

The Shim infrastructure, provided by Microsoft for backward compatibility, allows Microsoft to update system APIs while not breaking applications. It does so by allowing API calls to be redirected from Windows to an alternative code — the shim.

Windows comes with a Shim engine which checks a shim database for any applicable shims whenever it loads a binary. Malware can install their own shim database on to an affected program, and the Shim engine will load the malware's DLL whenever the program is run. The malware can then intercept any calls that the program makes.

Mitigation

By Developers

To mitigate against DLL injections, developers can hook into the LoadLibrary and CreateRemoteThread system calls. By hooking into LoadLibrary, developers can perform a library validation against a whitelist every time the function is called. If the DLL is on the whitelist, LoadLibrary will be allowed to proceed. For CreateRemoteThread, if the developer knows that he is not using that call, he can hook into it and disable the function's capabilities.

However, such a method is not completely foolproof, and can be more trouble than it is worth or impossible to implement. For example, if the application allows users to install plugins using DLLs like Outlook, it would be impossible for the developers to implement either a whitelist or a blacklist to LoadLibrary. Another example is an antivirus injecting itself into applications. If the developer implemented a whitelist, his application could be blocked by the antivirus from executing.

By System Administrators

As process injections are an integral part of the operating system, system administrators will not be able to completely mitigate against malware using process injection techniques specifically.

However, there are a few tools and techniques that can be considered to prevent and detect process injection situations. Here are four of them:

1. Install anti-malware with heuristics capabilities or endpoint detection and response (EDR) products. These products use API hooking to detect Windows API calls commonly used by malware authors. Combined with heuristics and machine learning, they have the capability to detect suspicious process injections and alert the user as it happens.

2. Whitelist applications using tools such as Microsoft's Applocker to aid system administrators in controlling what applications and files a user can execute. A carefully curated whitelist will prevent unvetted software from running. Also, as Applocker also controls execution of DLLs, it can prevent unknown injected DLLs from running. However, system administrators must note that this will incur a performance penalty as Applocker will need to check every DLL being loaded. One drawback of Applocker, however, is that it determines its actions based on the file name. If the malware's executable file is found in the whitelist (eg a malware might name itself "notepad.exe"), Applocker will allow it to execute.
3. Manage privileges and access using User Access Control (UAC). UAC is a built-in mechanism in Windows that helps to mitigate the impact of malware. System administrators should grant minimal privileges to users and disallow elevation of privileges without the administrator's consent. Any processes launched by a standard user would inherit the user's permissions and would be limited from making system level changes. This prevents malware from conducting unauthorised operations such as turning off the firewall or modifying registry settings.
4. Use exploit mitigation tools such as Microsoft's Arbitrary Code Guard (ACG). It is an exploit mitigation method that:

- Prevents a process from modifying existing executable process memory, and
- Prevents a process from allocating new executable memory without code written to disk.

ACG is a per-process configuration that system administrators can make to protect executables from process injection. However, in-depth testing must be conducted to ensure that the executable can still function properly, especially with EDR solutions. Also, while ACG makes it harder for malware to create executable code in memory using DLL injections, remote processes can still write to and execute shell code in an ACG enabled process.

Anti-malware tools with EDR and exploit mitigation tools such as ACG outlined above serve to prevent process injection as it happens. Both of them will actively stop process injection situations when they detect it. Applocker and UAC, which are both currently deployed in the GSIB environment, aid in mitigating the impact of malware and its persistency if one manages to slip through the net.

It is also important to note that process injection is transient; the malware process needs to run first before it can inject. In order to survive a reboot, the malware would need a means of running on system startup. Tight controls such as UAC and least privilege access controls would severely hamper its ability to do so.

Conclusion

Process injection is a mechanism that Windows and many of its applications depend on. While it was developed for legitimate purposes, it can be subverted by malware authors for nefarious purposes. Even though it is difficult to counter process injection techniques, defence in depth is still effective in countering the other stages of the malware's infection lifecycle. Disrupting any single stage in the malware's lifecycle would be enough to prevent the malware's operators from achieving their goal.

CSG @ GovTech

GovTech CSG—keeping the Singapore Government's ICT and Smart Systems safe and secure

Written by

Angelystor

CSG—cyber lead for the Singapore Government sector—keeping the Singapore Government's ICT and Smart Systems safe and secure. Our blog is all about the techniques and technologies in cybersecurity. We post fortnightly. Till then, stay cyber safe, and cyber ready!

More From Medium