ATTACK & DETECT ADVANCE PROCESS INJECTION TECHNIQUES MINI-COURSE

By- CyberWarFare Labs Team

Attack & Detect Advance Process Injection Techniques

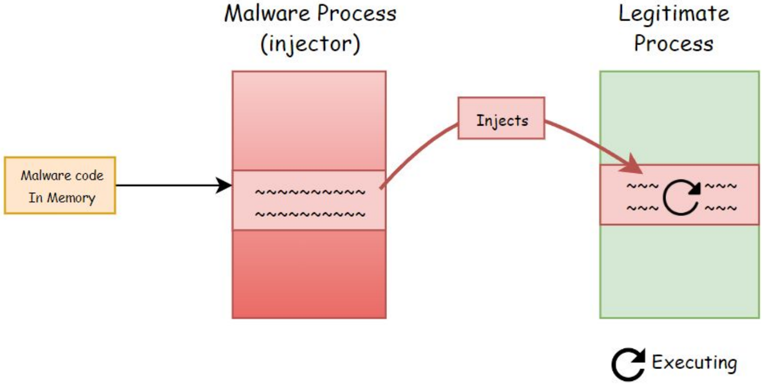
1. Process Injection Mindset
2. Classic Process Injection
3. APC Code Injection
4. Section Mapping
5. Module Stomping
6. Process Hollowing
7. Process Doppelganging
8. Transacted Hollowing
9. Process Herpaderping
10. Process Ghosting

PRE-REQUISITES

- Vmware/VirtualBox
- Windows 10 x64
  - Lab version: Windows 10 version 22H2 (x64bit)
- Any IDE or Editor (Visual studio 2022 is preferred)
- System Informer, PE-bear, CFF Explorer, x64dbg
- Microsoft ATP for Event Analysis or any event analysis environment that you prefer
  - Work Email is required to register in Microsoft ATP
- Programming Language:
  - *C/C++*

Basic Mindset for Process Injection (remote)

- Injecting PE/DLL/shellcode (malicious) into another process' address space
  - To hide from the AV products
  - To hide from the naked eye of the analyst
  - Sometimes, to access the resources (network, memory, files etc.) owned by another process
- When performing process injection, we need to have the following queries in our mind
  - How can we access the remote process?
  - How can we send our malicious code to the remote process?
  - How can we execute our malicious code which is inside the remote process?


Malware Process
(injector)
Legitimate
Process
Malware code
In Memory
Injects
~~~~~~~~~~~
~~~~~~~~~~~
Executing

Process Injection - Access Remote Process

- Getting Access to Remote Process
  - Obtain a handle to the remote process
    - Handle is value given to the user-mode processes when they try to access some object (process, thread, file, etc) from user-land
- Obtaining the handle
  - Opening a executing process
    - Win32 API: **OpenProcess**
    - NT API: **NtOpenProcess**
  - Creating a new legitimate process
    - Win32 API: **CreateProcessA**
    - NT API: **NtCreateProcessEx, NtCreateUserProcess**
  - Duplicating existing process handle from another process
    - Win32 API: **DuplicateHandle**
    - NT API: **NtDuplicateObject**

Process Injection - Sending Malicious Code

- Many ways to send malicious code to remote process, but few queries to have in our mind
  - Do we have enough privilege to write code into the remote process?
    - PROCSS_VM_OPERATION, PROCESS_VM_WRITE
  - Can we locate the address of the malicious code in the remote process that we just sent?
  - Is the memory region in remote process has enough memory access rights to write & execute code in that memory region?
    - Commonly we look for writable (W) & executable (X) memory region
      - But in modern OS because of security reason memory region is usually either writable or executable (W^X).

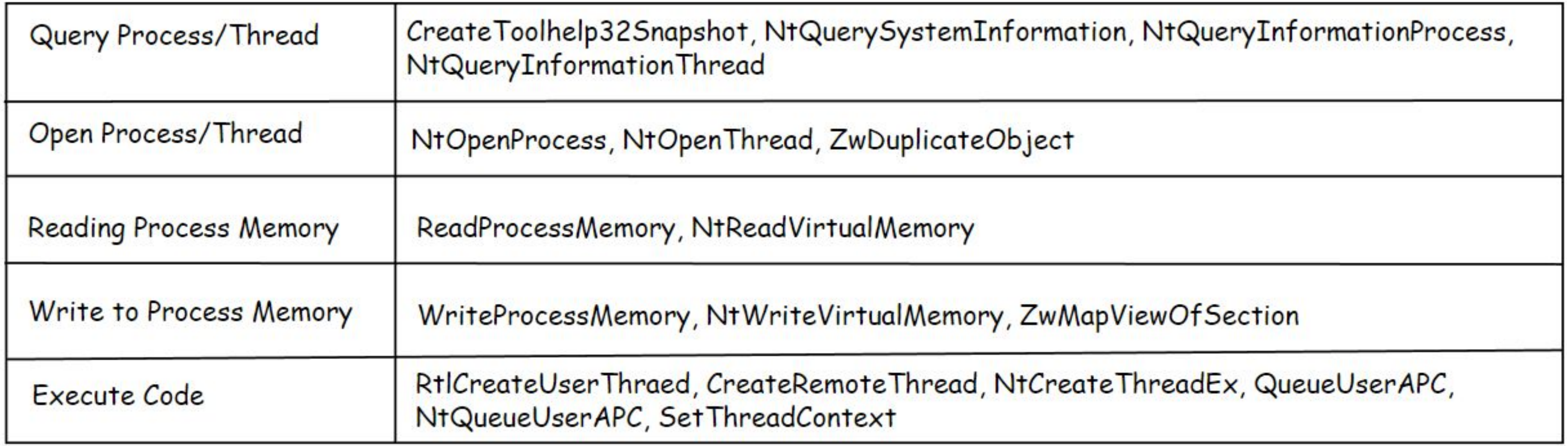
Process Injection - Sending Malicious Code

- Usually sending/injecting malicious code in remote process involves
  - Allocating new memory region with READ, WRITE & Execute access in remote process
    - Win32 API: **VirtualAllocEx**
    - NT API: **NtAllocateVirtualMemory**
    - Access Rights: PAGE_READWRITE, (PAGE_READWRITE | PAGE_EXECUTE)
  - Writing payload into the memory
    - Win32 API: **WriteProcessMemory**
    - NT API: **NtWriteVirtualMemory**
- Additionally, changing memory protection also involves in this stage
  - Usually, memory protection PAGE_READWRITE is changed to PAGE_EXECUTE_READ and vice versa.
    - Win32 API: **VirtualProtectEx**
    - NtAPI: **NtProtectVirtualMemory**

Process Injection - Execute the Malicious code

- Common ways to perform execution
  - Create a new thread in target process
    - Win32 API: **CreateRemoteThread**
    - NT API: **NtCreateThreadEx**, **RtlCreateUserThread**
  - Queuing APC in alertable thread
    - Win32 API: **QueueUserAPC**
    - NT API: **NtQueueUserAPC**
  - Hijacking the executing thread
    - Win32 API: **SetThreadContext**
    - NT API: **NtSetContextThread**
- Last phase of the injection
  - Some APIs that are used in this stage are heavily monitored by the AV/EDR products

Process Injection - Common APIs

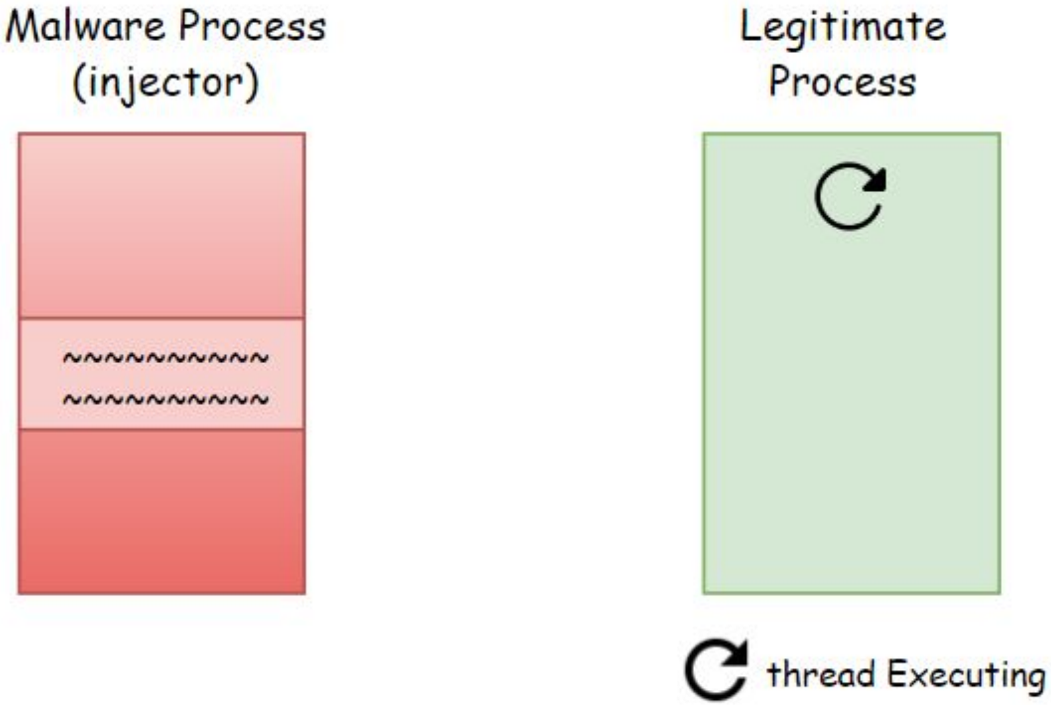


Query Process/Thread	CreateToolhelp32Snapshot, NtQuerySystemInformation, NtQueryInformationProcess, NtQueryInformationThread
Open Process/Thread	NtOpenProcess, NtOpenThread, ZwDuplicateObject
Reading Process Memory	ReadProcessMemory, NtReadVirtualMemory
Write to Process Memory	WriteProcessMemory, NtWriteVirtualMemory, ZwMapViewOfSection
Execute Code	RtlCreateUserThraed, CreateRemoteThread, NtCreateThreadEx, QueueUserAPC, NtQueueUserAPC, SetThreadContext

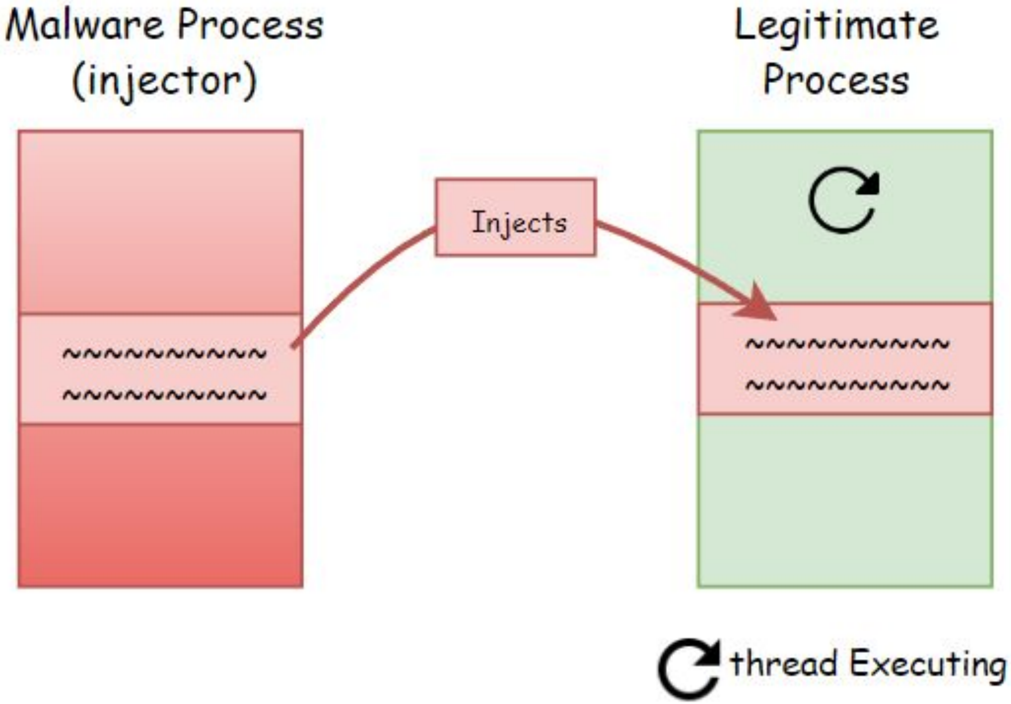
Classic Process Injection - steps

- Obtain Handle to a target process
  - CreateToolHelp32Snapshot, OpenProcess, NtQuerySystemInformation
- Allocate new memory region at target process
  - VirtualAllocEx, NtAllocateVirtualMemory
- Write payload into newly allocated memory
  - WriteProcessMemory, NtWriteVirtualMemory
- Create new remote thread
  - CreateRemoteThread, NtCreateThreadEx

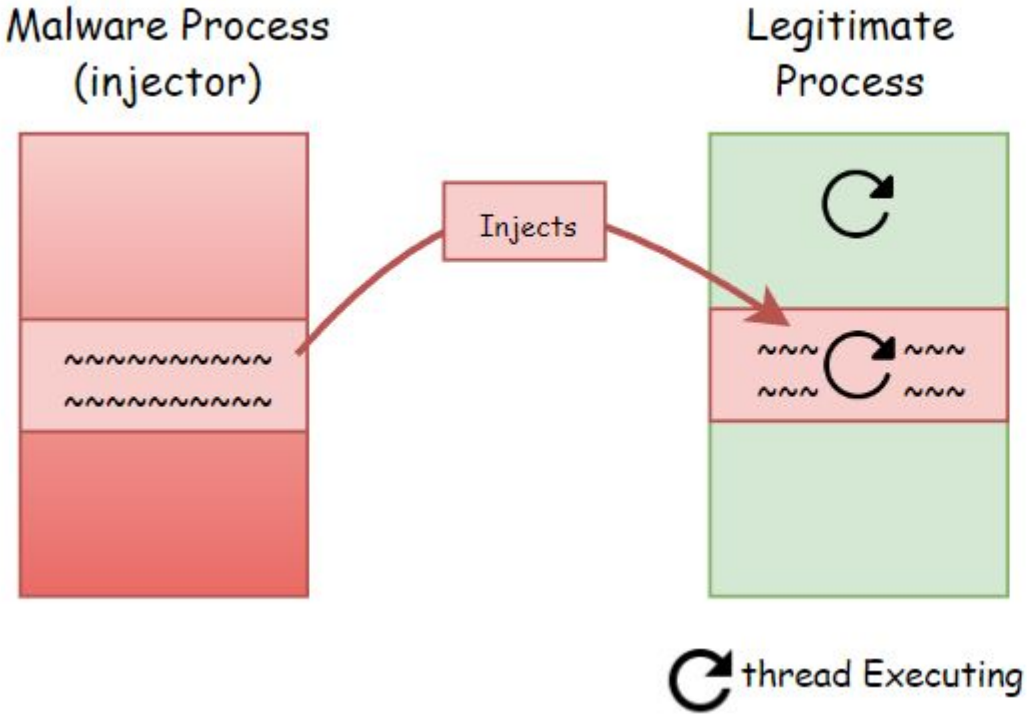
Classic Process Injection


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Legitimate
Process
thread Executing

Classic Process Injection


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
Injects
Legitimate
Process
~~~~~~~~~~
~~~~~~~~~~
thread Executing

Classic Process Injection


Malware Process (injector)
~~~~~~~~~~~
~~~~~~~~~~~
Injects
Legitimate Process
~~~ ~~~
~~~ ~~~
thread Executing

Classic Process Injection - API calls

- Kernel32.dll :
  - *CreateToolHelp32Snapshot, Process32First, Process32Next, Thread32First, Thread32Next, OpenProcess, WriteProcessMemory, VirtualProtectEx, OpenThread*
- Ntdll.dll:
  - *NtQuerySystemInformation,NtAllocateVirtualMemory, NtWriteVirtualMemory*

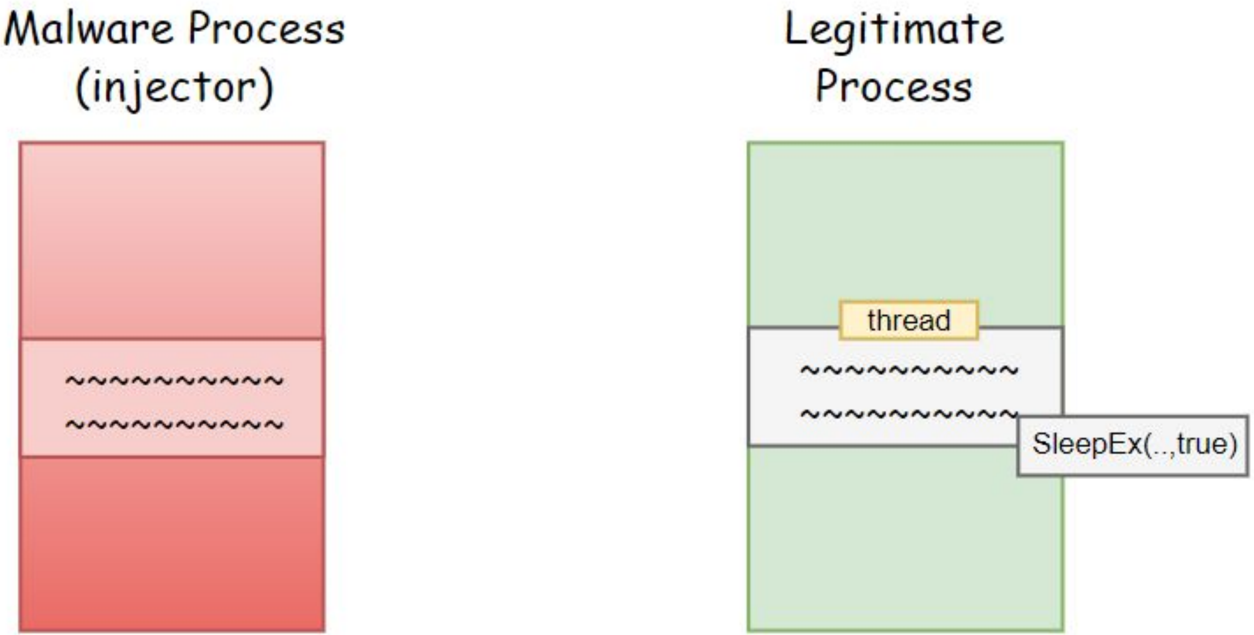
APC Code Injection

- APC stands for Asynchronous Procedure Call
- APC functions execute asynchronously in context of a particular thread
- In this techniques our shellcode is placed in APC Queue of the thread.
- The payload will get executed when the thread goes to alertable state
- Wait routines puts thread in alertable state, such as:
  - *SleepEx()*
  - *WaitForSingleObjectEx()*
  - *WaitForMultipleObjectEx()*

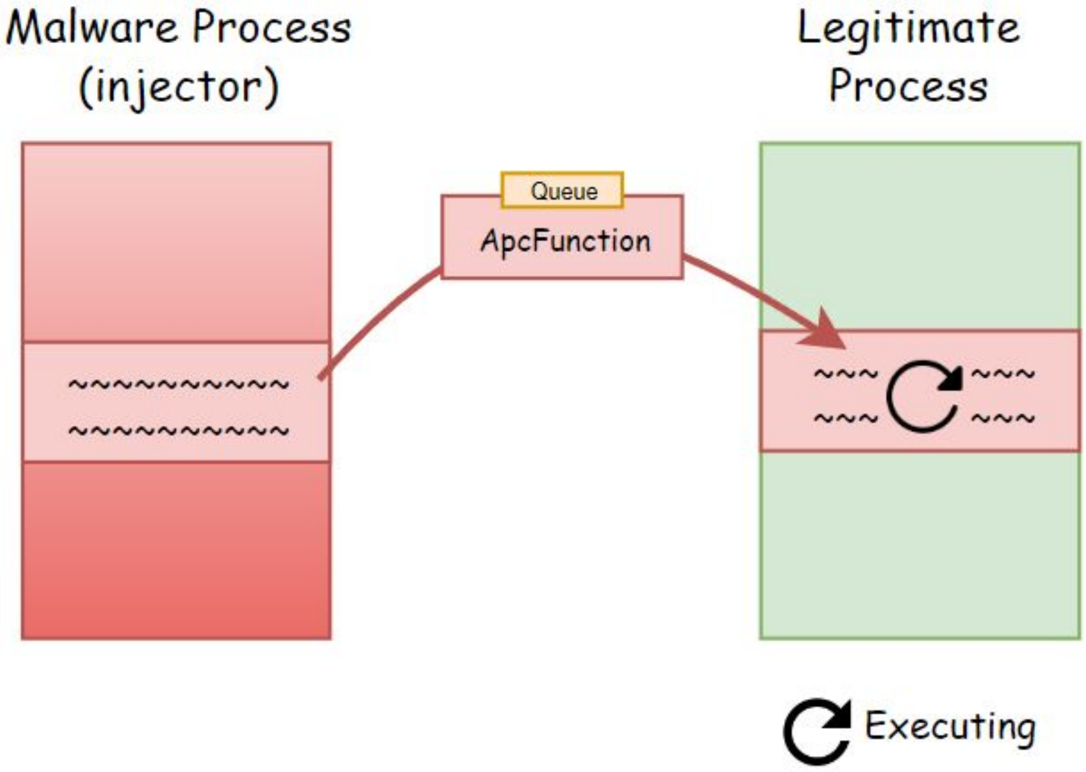
APC Code Injection - Steps

- Find the process to inject our payload
  - *CreateToolHelp32Snapshot, NtQuerySystemInformation*
- Find all the threads in that process
  - *Thread32First, Thread32Next*
- Allocate memory in that process
  - *VirtualAllocEx, NtAllocateVirtualMemory*
- Write the payload into that allocated memory
  - *WriteProcessMemory, NtWriteVirtualMemory*
- Put the APC function in the queue for all threads
  - *QueueUserAPC, NtQueueUserAPC*
- APC function here points to our shellcode

APC Code Injection


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Legitimate
Process
thread
~~~~~~~~~~~
~~~~~~~~~~~
SleepEx(..,true)

APC Code Injection


Malware Process (injector)
~~~~~~~~~~~
~~~~~~~~~~~
Queue
ApcFunction
Legitimate Process
~~~ ~~~
~~~ ~~~
Executing

APC Code Injection - API calls

- Kernel32.dll :
  - *CreateToolHelp32Snapshot, Process32First, Process32Next, Thread32First, Thread32Next, OpenProcess, WriteProcessMemory, VirtualProtectEx, OpenThread, QueueUserAPC*
- Ntdll.dll:
  - *NtQuerySystemInformation,NtAllocateVirtualMemory, NtWriteVirtualMemory*

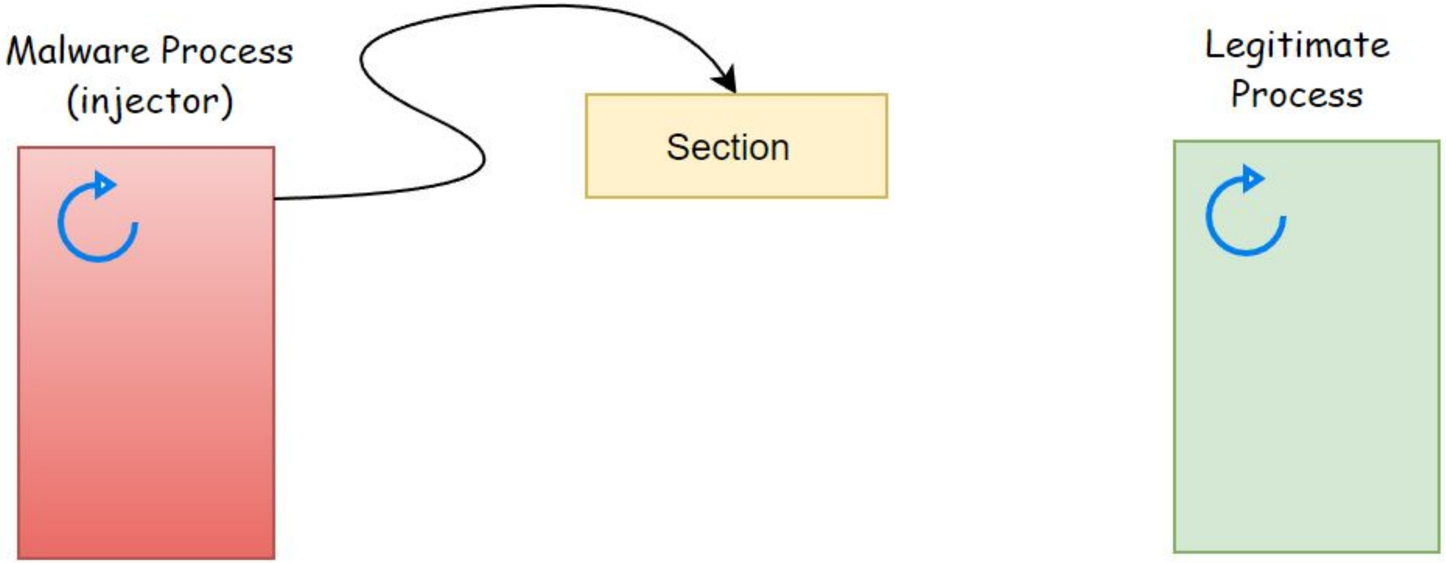
Section Mapping

- Block of memory that can be shared between multiple processes [1]
- In memory, each section has corresponding views, which are parts of the section that are visible to processes.
    - Act of creating a view for a section is known as mapping a view of the section [1]
- In this technique a section is created and view of section is mapped to both local & target process with different page protection

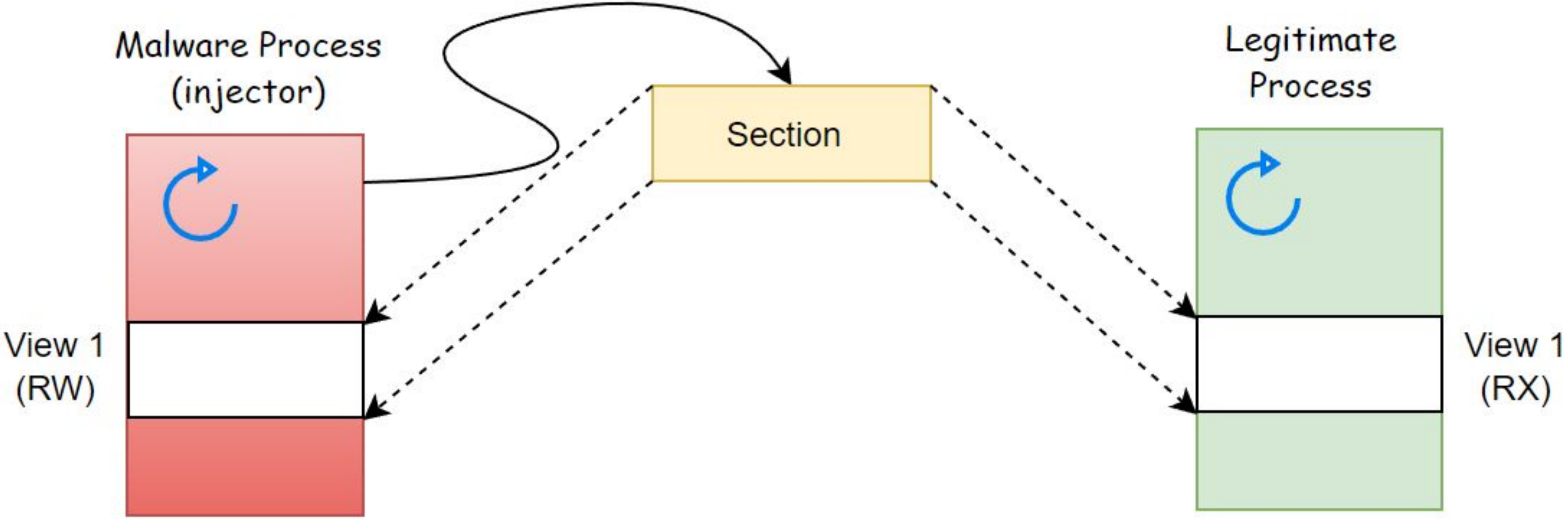
Section Mapping - Steps

- Create a new section with full RWX page protection
  - NtCreateSection
- Map a view of section to local process (injector) with RW page protection
  - NtMapViewOfSection
- Map a view of section to target process with RX page protection
  - NtMapViewOfSection
- Write a payload to a view mapped to a local process
  - memcpy
- Create a remote thread with a base address of view mapped to remote process
  - CreateRemoteThread, NtCreateThreadEx, RtlCreateUserThread

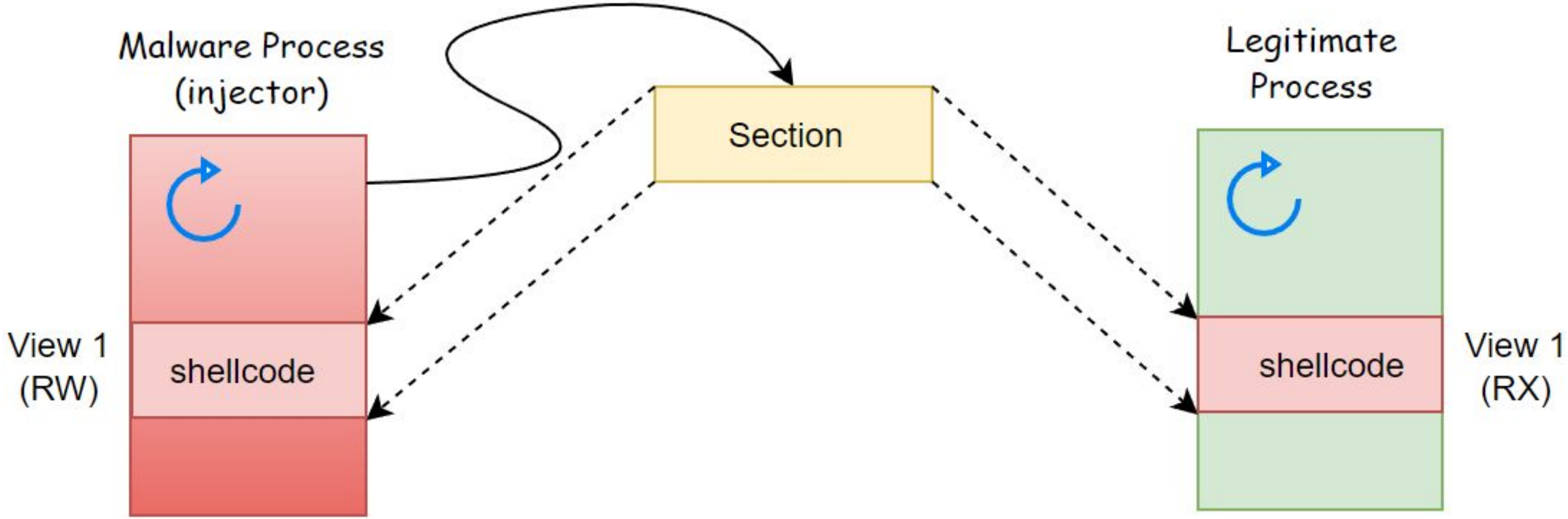
Section Mapping



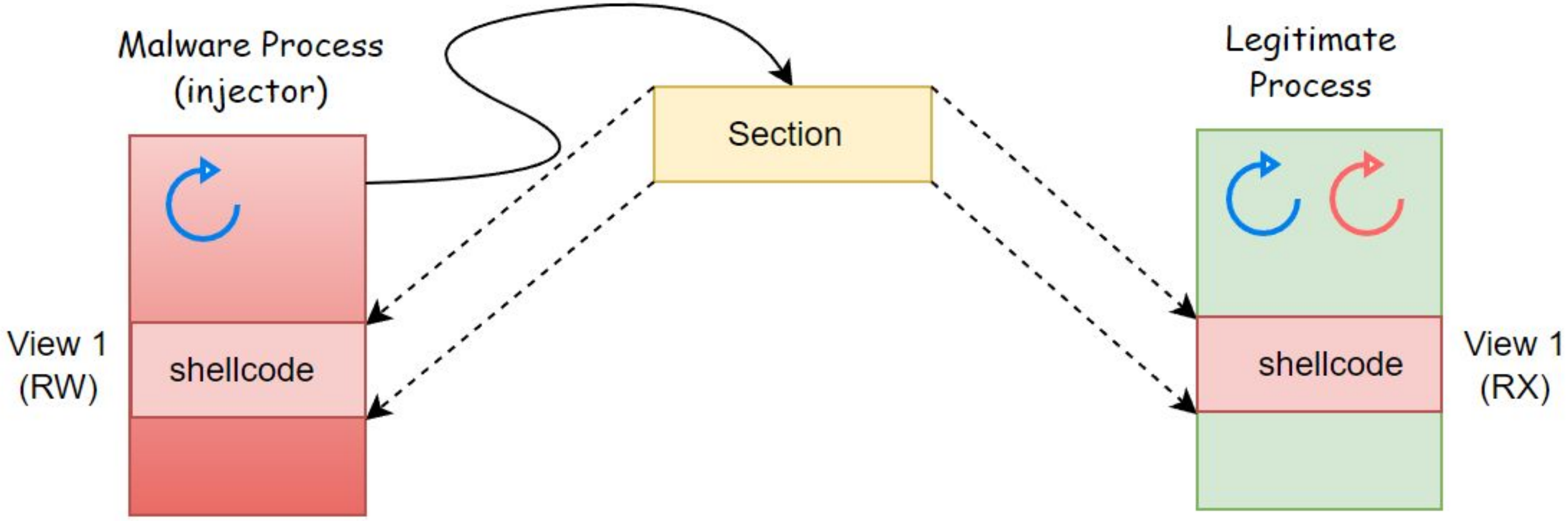
Section Mapping



Section Mapping



Section Mapping



Section Mapping - API calls

- Kernel32.dll :
  - *OpenProcess, CreateRemoteThread*
- Ntdll.dll:
  - *NtCreateSection, NtMapViewOfSection, NtCreateThreadEx*

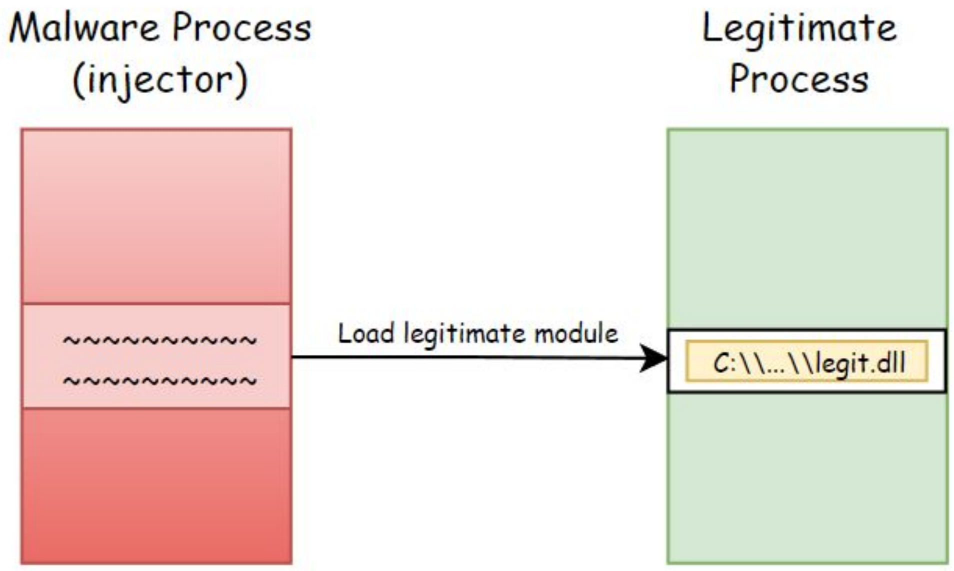
Module Stomping

- ➔ This is the technique to load fresh dll into the target process and inject the shellcode into it
- ➔ No need to change memory protection in target process memory
- ➔ Shellcode gets executed from the legitimate dll

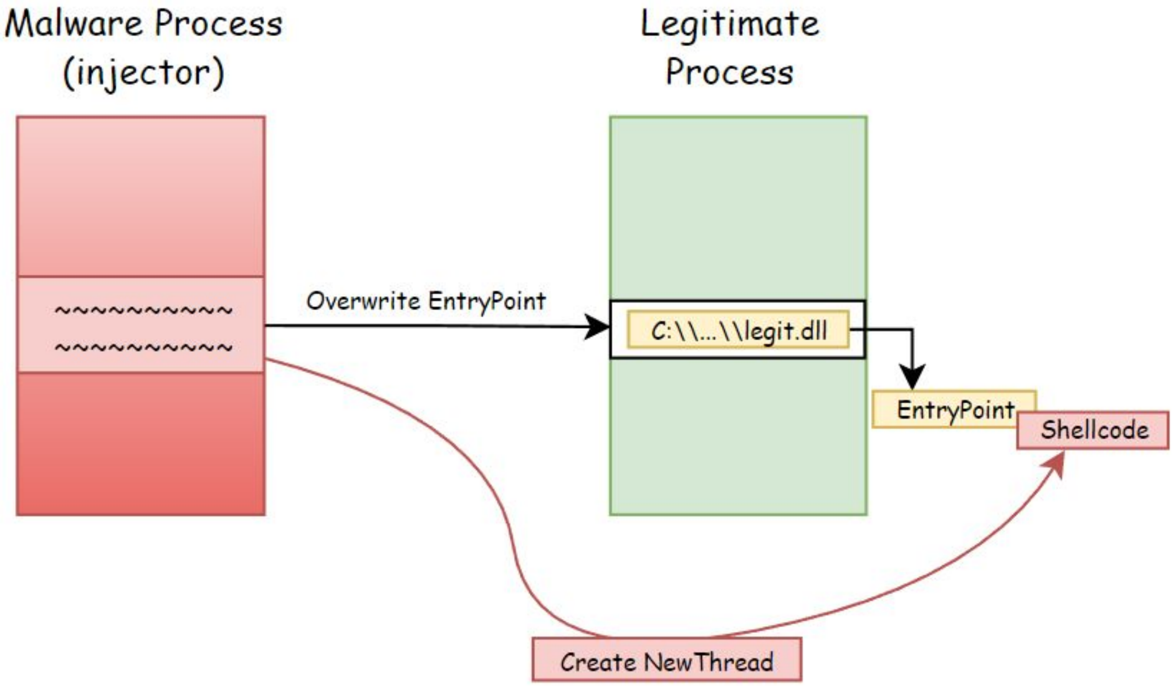
Module Stomping - Steps

- Open a target process and get handle to the target process
  - *OpenProcess, NtOpenProcess*
- Load the target module in the target process
  - *VirtualAllocEx, WriteProcessMemory, CreateRemoteThread*
- Write the payload at the entrypoint address of the loaded module
  - *WriteProcessMemory*
- Create a thread to execute the payload
  - *CreateRemoteThread*

Module Stomping


Malware Process (injector)
~~~~~~~~~~~
~~~~~~~~~~~
Load legitimate module
Legitimate Process
C:\\...\\legit.dll

Module Stomping


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Overwrite EntryPoint
Legitimate
Process
C:\\...\\legit.dll
EntryPoint
Shellcode
Create NewThread

Module Stomping - API calls

- Kernel32.dll :
  - *OpenProcess, ReadProcessMemory, WriteProcessMemory, VirtualAllocEx, VirtualProtectEx, CreateRemoteThread*
- Psapi.dll:
  - *EnumProcessModules, GetModuleFileNameEx*
- Ntdll.dll:
  - *NtAllocateVirtualMemory*

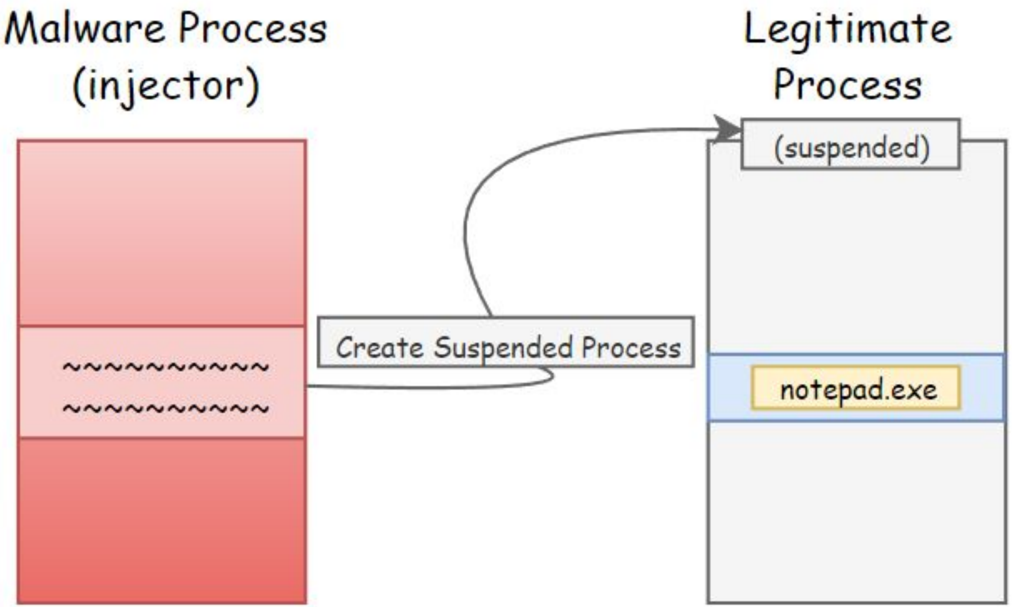
Process Hollowing

- ➔ Replace executable section of the legitimate process with malicious executable.
- ➔ Replacement takes place in memory.
- ➔ Malicious code executes from inside of the legitimate process thus, it conceals its presence.
- ➔ The path of the hollowed process still points to the legitimate executable path.

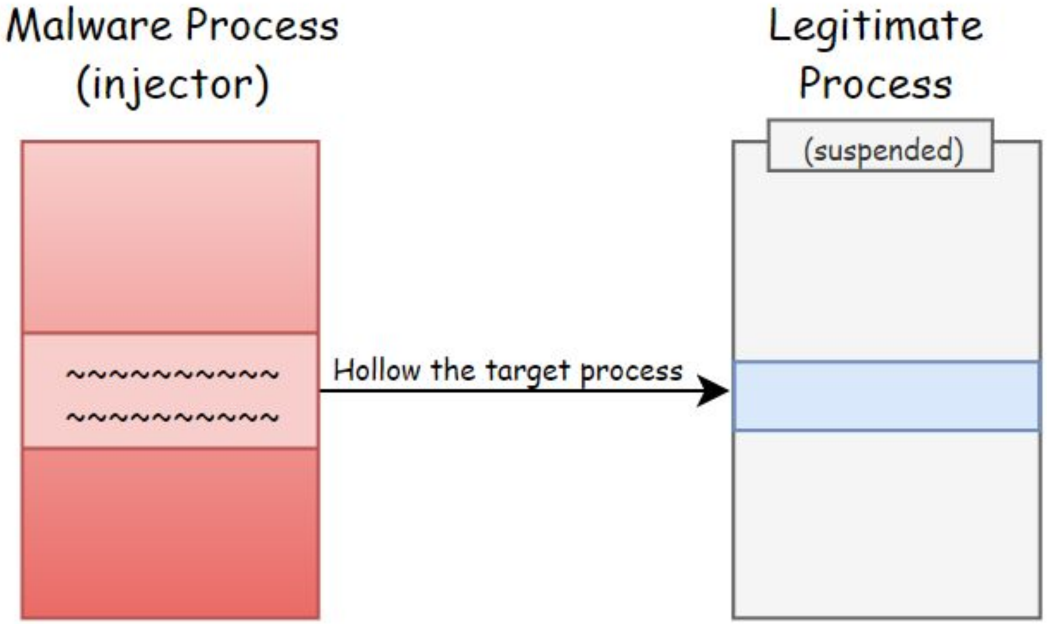
Process Hollowing - Steps

- Create target process in suspended mode
  - *CreateProcessA*
- Get Image Base Address of the target process
  - *NtQueryInformationProcess, ReadProcessMemory*
- Hollow/Unmap target image
  - *ZwUnmapViewOfSection*
- Allocate new memory in target process for the payload
  - *VirtualAllocEx*
- Copy all the payload section to the allocated memory in target process
  - *WriteProcessMemory*
- Get Context of target process
  - *GetThreadContext*
- Set the entrypoint of payload in respective context
  - *EAX for x86, RCX for x64*
- Apply the Context of target process
  - *SetThreadContext*
- Resume main thread of target process
  - *ResumeThread*

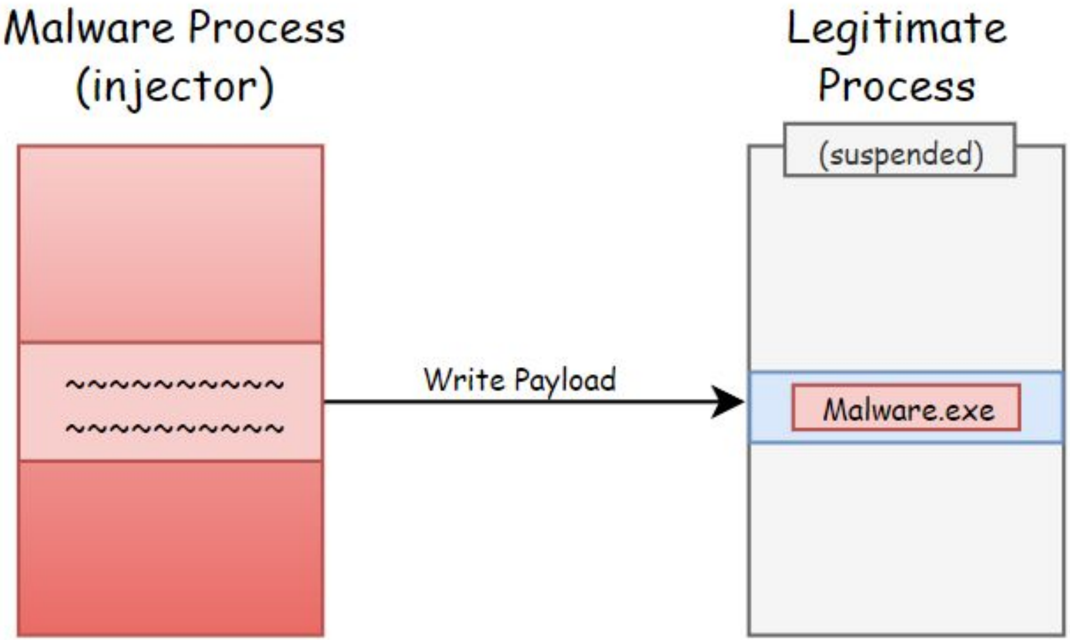
Process Hollowing


Malware Process (injector)
~~~~~~~~~~~
~~~~~~~~~~~
Create Suspended Process
Legitimate Process
(suspended)
notepad.exe

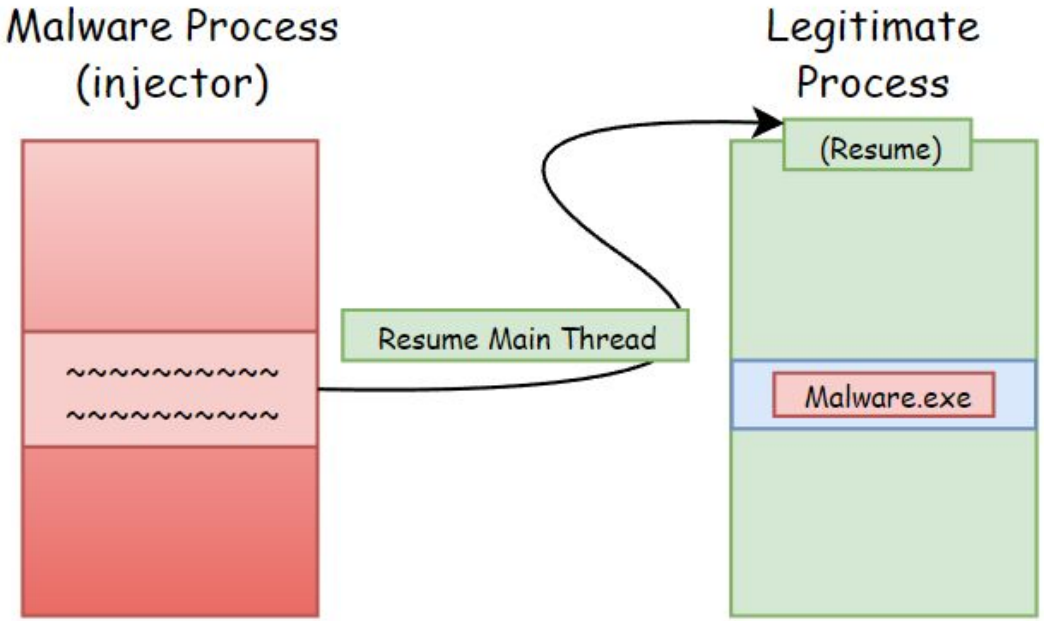
Process Hollowing


Malware Process (injector)
~~~~~~~~~~~
~~~~~~~~~~~
Hollow the target process
Legitimate Process
(suspended)

Process Hollowing


Malware Process (injector)
~~~~~~~~~~~
~~~~~~~~~~~
Write Payload
Legitimate Process
(suspended)
Malware.exe

Process Hollowing


Malware Process (injector)
~~~~~~~~~~~
~~~~~~~~~~~
Resume Main Thread
Legitimate Process
(Resume)
Malware.exe

Process Hollowing - API Calls

- Kernel32.dll:
  - CreateProcessA, ReadProcessMemory, WriteProcessMemory, GetThreadContext, SetThreadContext, ResumeThread
- Ntdll.dll:
  - NtQueryInformationProcess, NtUnmapViewOfSection/ZwUnmapViewOfSection

Process Doppelganging

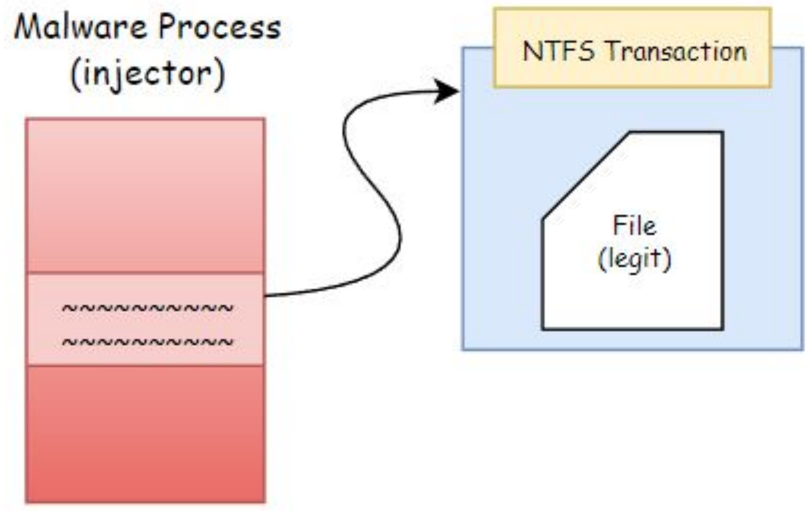
- Process Doppelganging utilizes the Windows API calls related to the NTFS transactions.
- Transactional NTFS brings the concept of atomic transactions to NTFS file system, which allows app developers and administrators to handle mistakes and maintain data integrity more easily.
- Transactional NTFS allows for files and directories to be created, modified, renamed and deleted atomically.
- In a series of file operations (performed in a transaction), when all operations complete successfully, the operation is committed. If an error occurs, the entire operation is rolled back and fails. This is to preserve integrity of data on disk.

Process Doppelganging - Steps

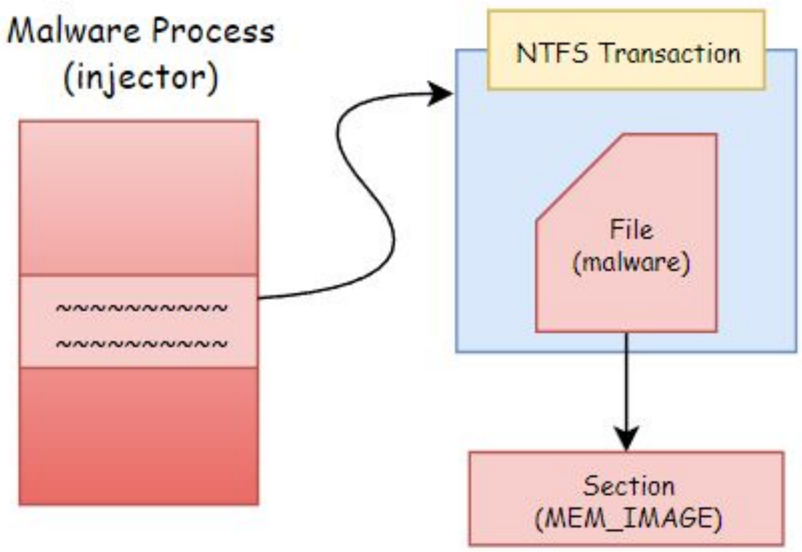
➔ Steps of Doppelganging can be broken down into 4 steps:

1. Transact : process a legitimate file into the NTFS transaction and then overwrite it with a malicious payload file
    - *CreateTransaction, CreateFileTransactedA*
2. Load: Create a memory section from the payload and load the malicious code
    - *NtCreateSection*
3. Rollback: Rollback the transaction i.e., removing malicious code so that no data left on the disk
    - *RollbackTransaction*
4. Animate: Bringing Doppelganging to life. Create a process from the previously created memory section (step 2). The memory section contains malicious code and never written to the disk.
    - *NtCreateProcessEx, NtCreateThreadEx*

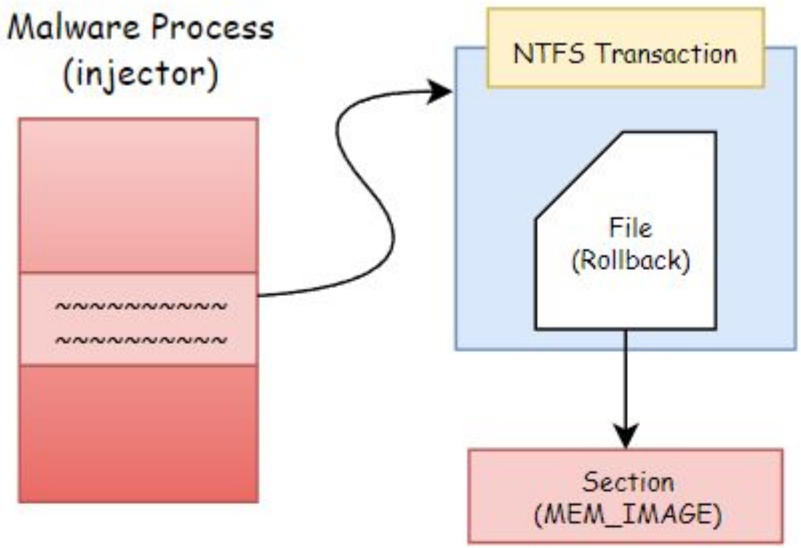
Process Doppelganging


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(legit)

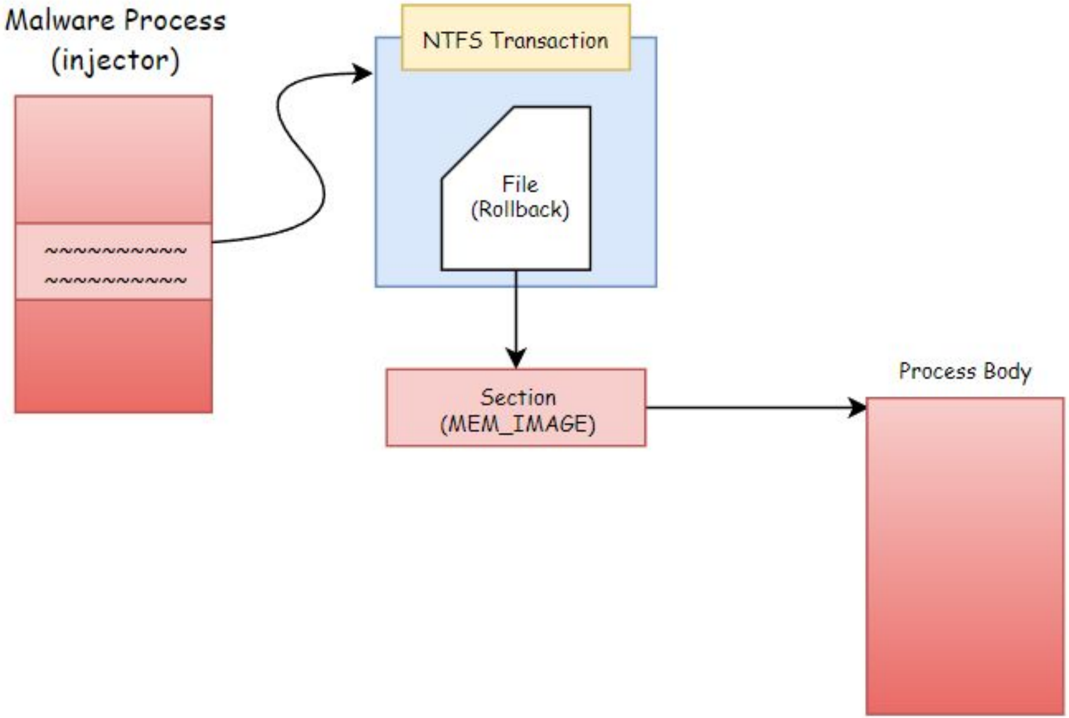
Process Doppelganging


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(malware)
Section
(MEM_IMAGE)

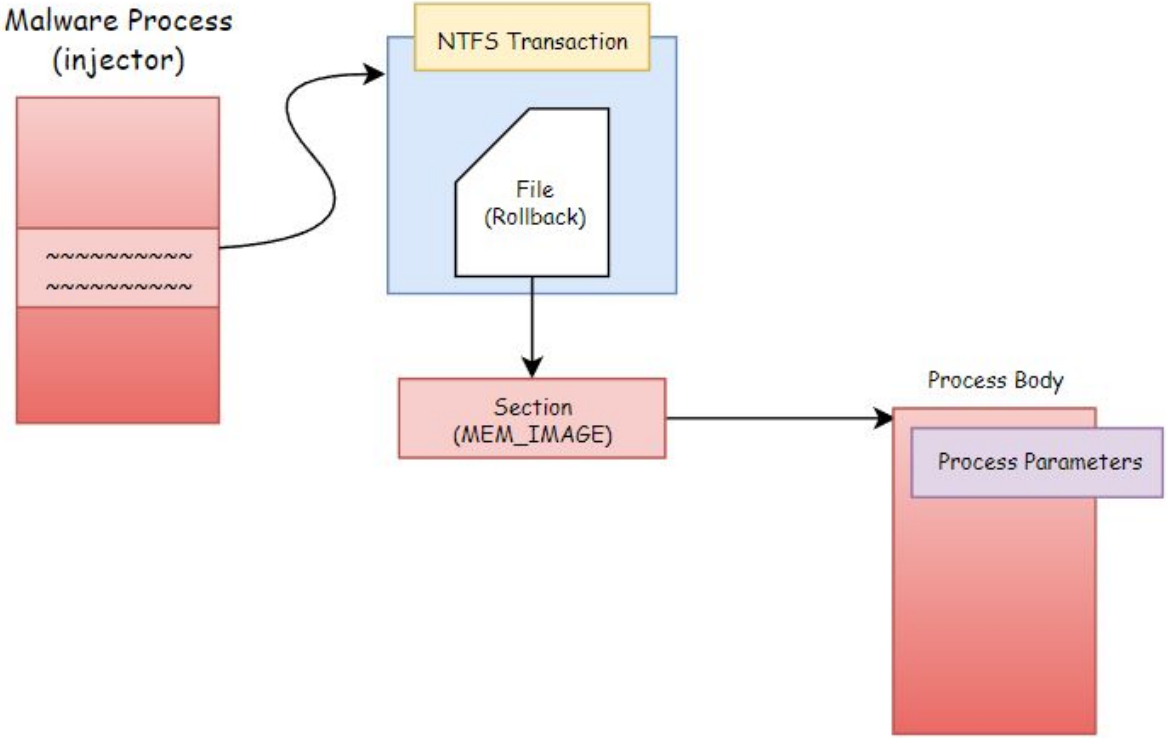
Process Doppelganging


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(Rollback)
Section
(MEM_IMAGE)

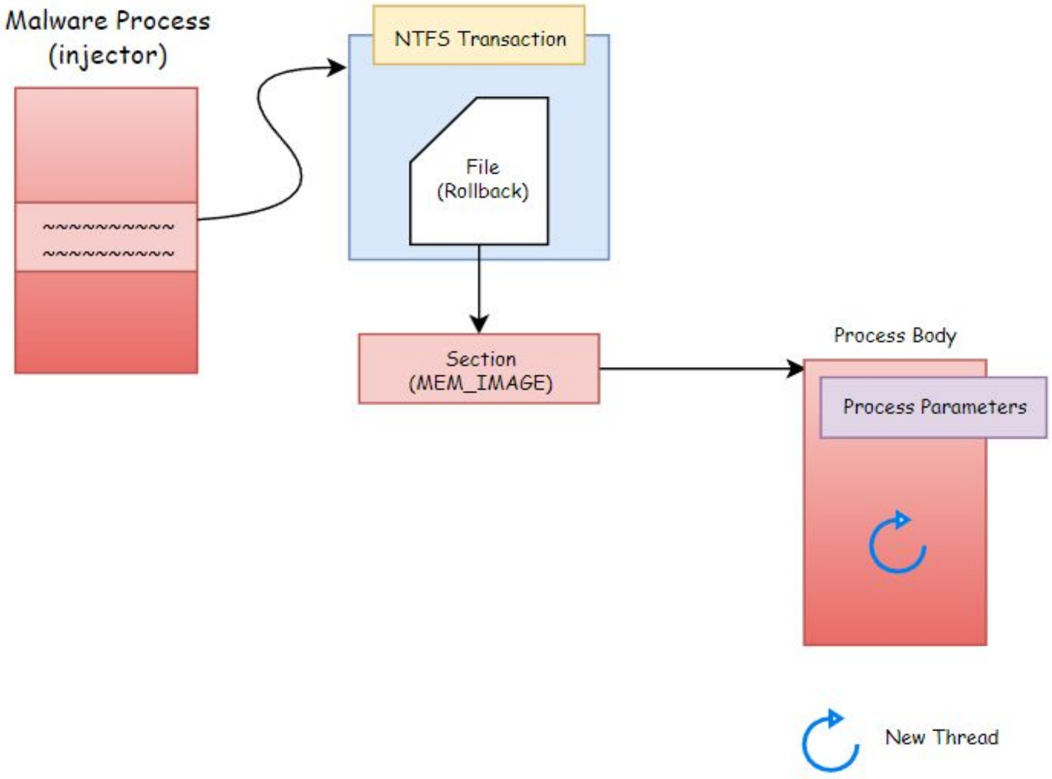
Process Doppelganging


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(Rollback)
Section
(MEM_IMAGE)
Process Body

Process Doppelganging


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(Rollback)
Section
(MEM_IMAGE)
Process Body
Process Parameters

Process Doppelganging



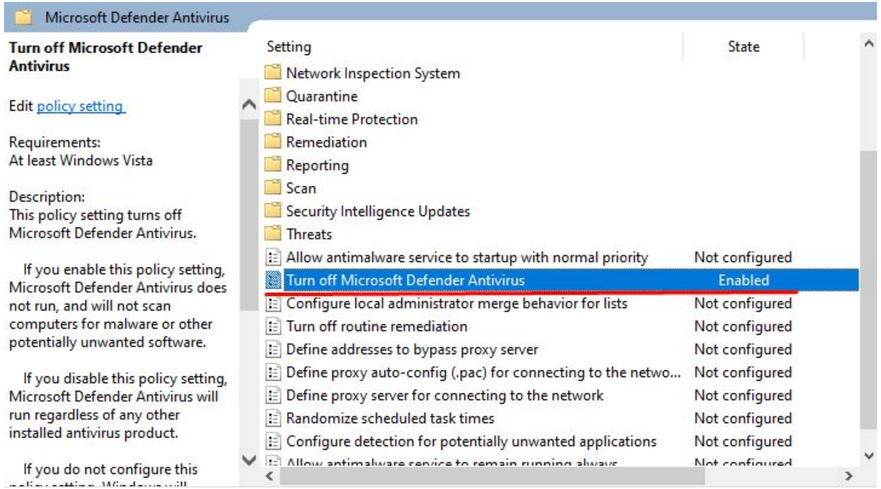
Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(Rollback)
Section
(MEM_IMAGE)
Process Body
Process Parameters
New Thread

Process Doppelganging - API Calls

- KtmW32.dll:
  - *CreateTransaction, RollbackTransaction*
- Kernel32.dll:
  - *CreateFileTransactedA, WriteFile*
- Ntdll.dll:
  - *NtCreateSection, NtCreateProcessEx, NtCreateThreadEx*

Process Doppelganging - Issue

- ➔ Microsoft Windows Defender is monitoring the creation of remote thread using the routine PsSetCreateThreadNotifyRoutine.
- ➔ Disable Microsoft Windows Defender from Group policy.
- ➔ Computer Configuration > Administrative Templates > Windows Components > Windows Defender Antivirus > Turn off Microsoft Defender Antivirus (Enabled)



Microsoft Defender Antivirus

Turn off Microsoft Defender Antivirus

Edit policy setting

Requirements:
At least Windows Vista

Description:
This policy setting turns off Microsoft Defender Antivirus.

If you enable this policy setting, Microsoft Defender Antivirus does not run, and will not scan computers for malware or other potentially unwanted software.

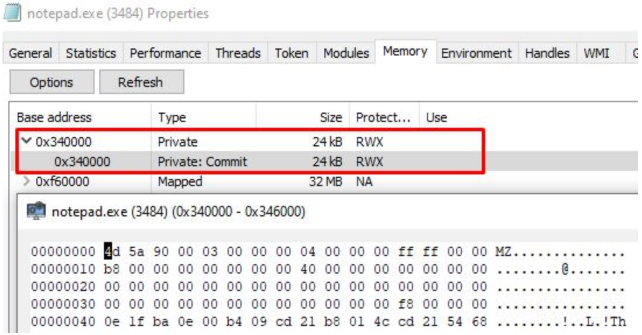
If you disable this policy setting, Microsoft Defender Antivirus will run regardless of any other installed antivirus product.

If you do not configure this policy setting, Windows will

Setting	State
Network Inspection System	
Quarantine	
Real-time Protection	
Remediation	
Reporting	
Scan	
Security Intelligence Updates	
Threats	
Allow antimalware service to startup with normal priority	Not configured
Turn off Microsoft Defender Antivirus	Enabled
Configure local administrator merge behavior for lists	Not configured
Turn off routine remediation	Not configured
Define addresses to bypass proxy server	Not configured
Define proxy auto-config (.pac) for connecting to the netwo...	Not configured
Define proxy server for connecting to the network	Not configured
Randomize scheduled task times	Not configured
Configure detection for potentially unwanted applications	Not configured
Allow antimalware service to remain running always	Not configured

Transacted Hollowing

- ➔ Hybrid of Process Hollowing and Process Doppelganging
- ➔ This technique solves the issues of both techniques

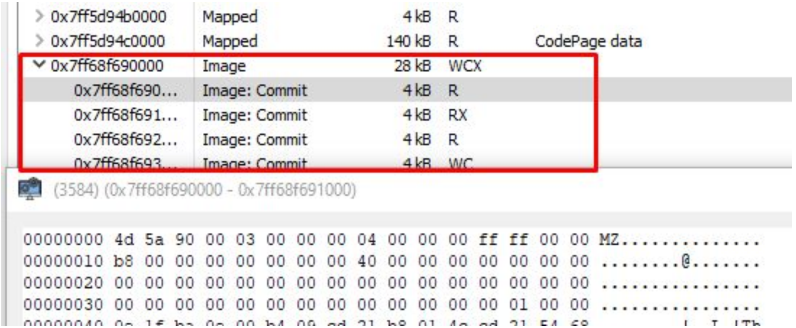


Base address	Type	Size	Protect...	Use
0x340000	Private	24 kB	RWX	
0x340000	Private: Commit	24 kB	RWX	
0xf60000	Mapped	32 MB	NA	

notepad.exe (3484) (0x340000 - 0x346000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 f8 00 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
```

Process Hollowing



0x7ff5d94b0000	Mapped	4 kB	R	
0x7ff5d94c0000	Mapped	140 kB	R	CodePage data
0x7ff68f690000	Image	28 kB	WCX	
0x7ff68f690...	Image: Commit	4 kB	R	
0x7ff68f691...	Image: Commit	4 kB	RX	
0x7ff68f692...	Image: Commit	4 kB	R	
0x7ff68f693...	Image: Commit	4 kB	WC	

(3584) (0x7ff68f690000 - 0x7ff68f691000)

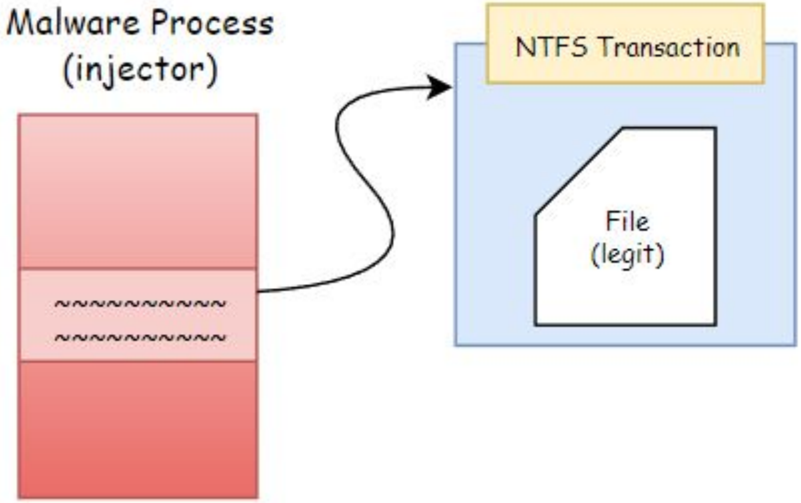
```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
```

Normal/ Process Doppelganging

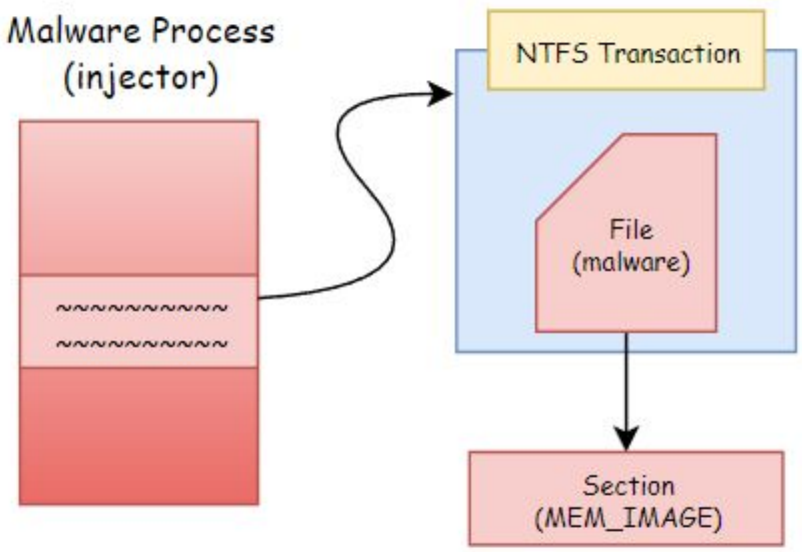
Transacted Hollowing - Steps

- Create NTFS transaction object
  - *CreateTransaction*
- Open/Create target file for transaction
  - *CreateFileTransactedA*
- Create an image section from transacted file
  - *NtCreateSection*
- Rollback the transaction
  - *RollbackTransaction*
- Create a new target process in suspended mode
  - *CreateProcessA*
- Map an image section into the target process
  - *NtMapViewOfSection*
- Update entrypoint in target process with payload entrypoint
  - *GetThreadContext, SetThreadContext*
- Update image base address at target process PEB with newly mapped image base address
  - *NtQueryInformationProcess, WriteProcessMemory*
- Resume the thread
  - *NtResumeThread*

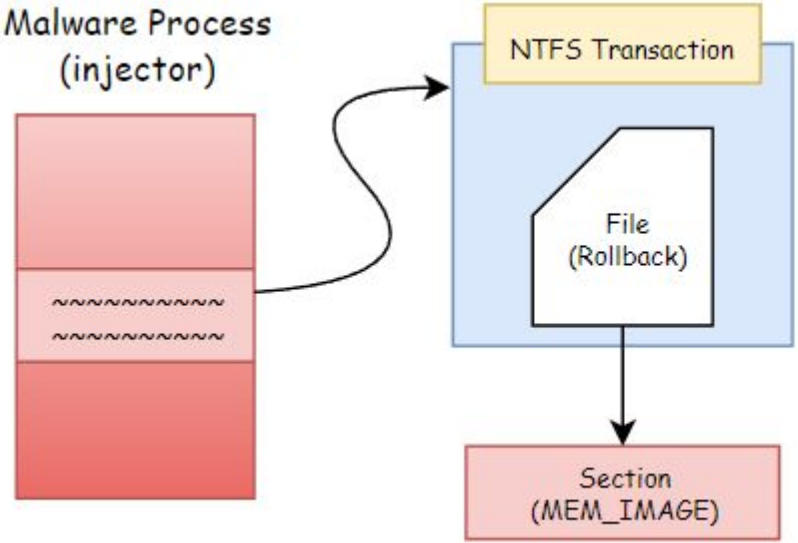
Transacted Hollowing


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(legit)

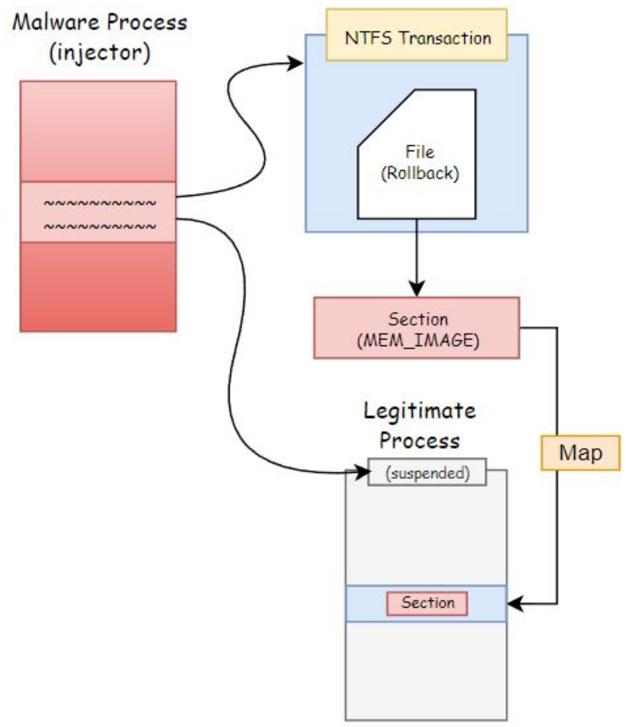
Transacted Hollowing


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(malware)
Section
(MEM_IMAGE)

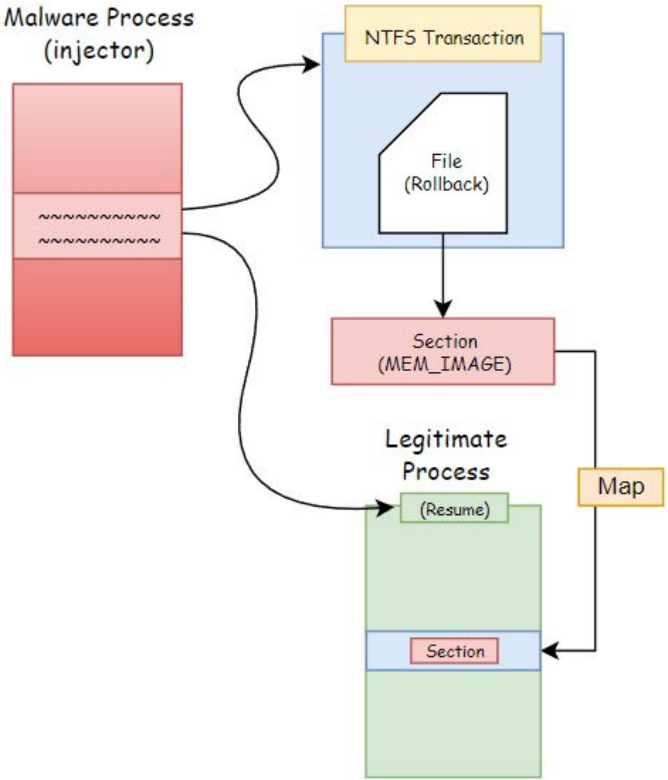
Transacted Hollowing


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(Rollback)
Section
(MEM_IMAGE)

Transacted Hollowing


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(Rollback)
Section
(MEM_IMAGE)
Map
Legitimate
Process
(suspended)
Section

Transacted Hollowing


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
NTFS Transaction
File
(Rollback)
Section
(MEM_IMAGE)
Map
Legitimate
Process
(Resume)
Section

Transacted Hollowing - API Calls

- Kernel32.dll:
  - CreateFileTransactedW, WriteFile, CreateProcessW, ResumeThread, GetThreadContext, SetThreadContext, ResumeThread
- Ntdll.dll:
  - NtQueryInformationProcess, NtCreateTransaction, NtCreateSection, NtRollbackTransaction, NtMapViewOfSection

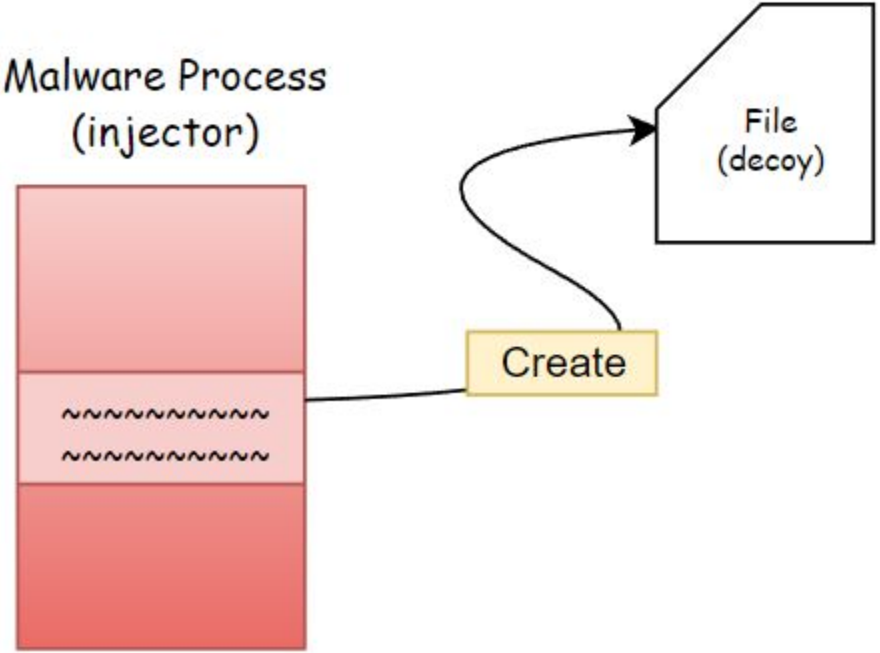
Process Herpaderping

- In this technique the file on-disk is modified after the image has been mapped
- The modification is done before creating an initial thread
- Temporary file on-disk act as a decoy
- At this point the file on-disk is different from the one executed in-memory

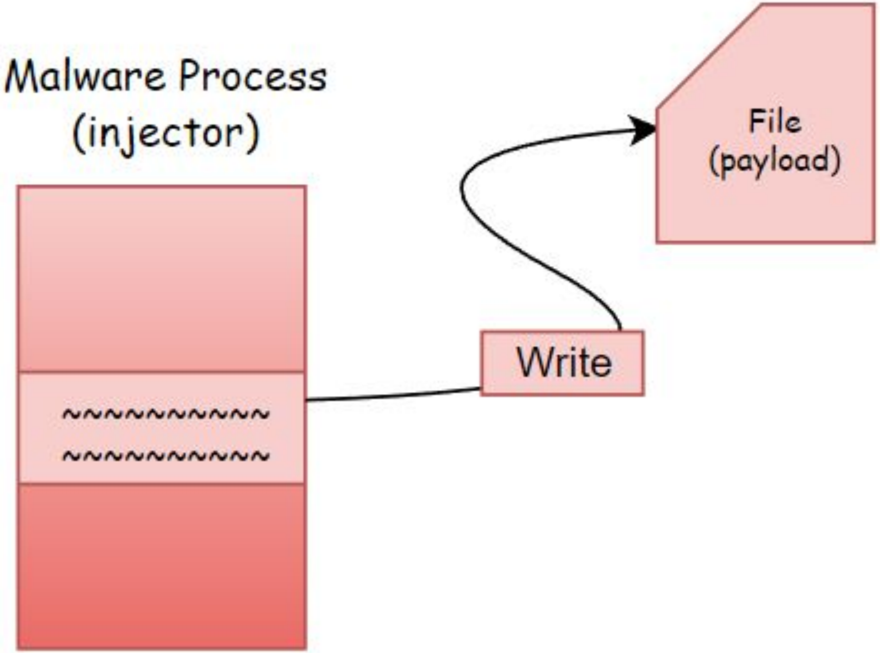
Process Herpaderping - Steps

- Create a temp/decoy file
  - *CreateFileA*
- Write payload into that file (do not close the temp file handle after writing payload into it)
  - *WriteFile*
- Create an image section from that file
  - *NtCreateSection*
- Create a process using the newly created section
  - *NtCreateProcessEx*
- Modify the temp file
  - SetFilePointer, WriteFile
- Setup process parameters
  - *RtlCreateProcessParametersEx*
- Create new thread
  - *NtCreateThreadEx*
- Close temp file handle
  - *CloseHandle*

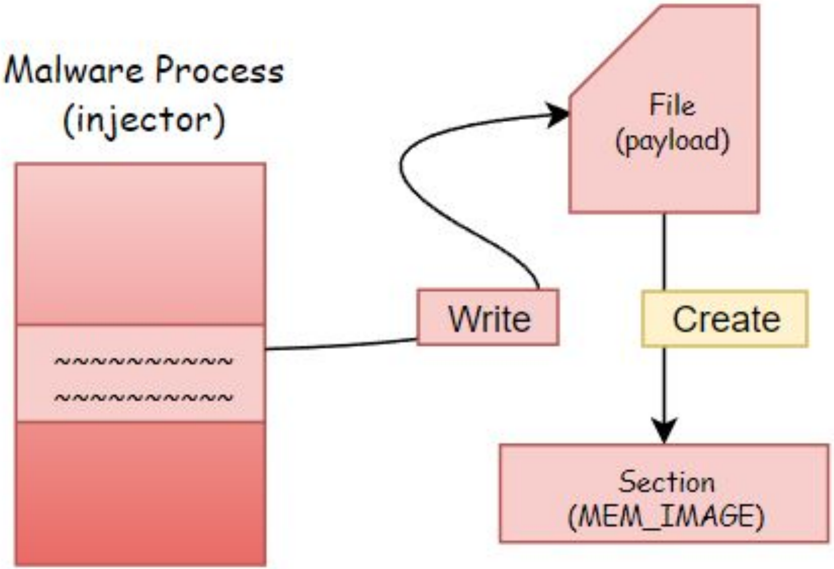
Process Herpaderping


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Create
File
(decoy)

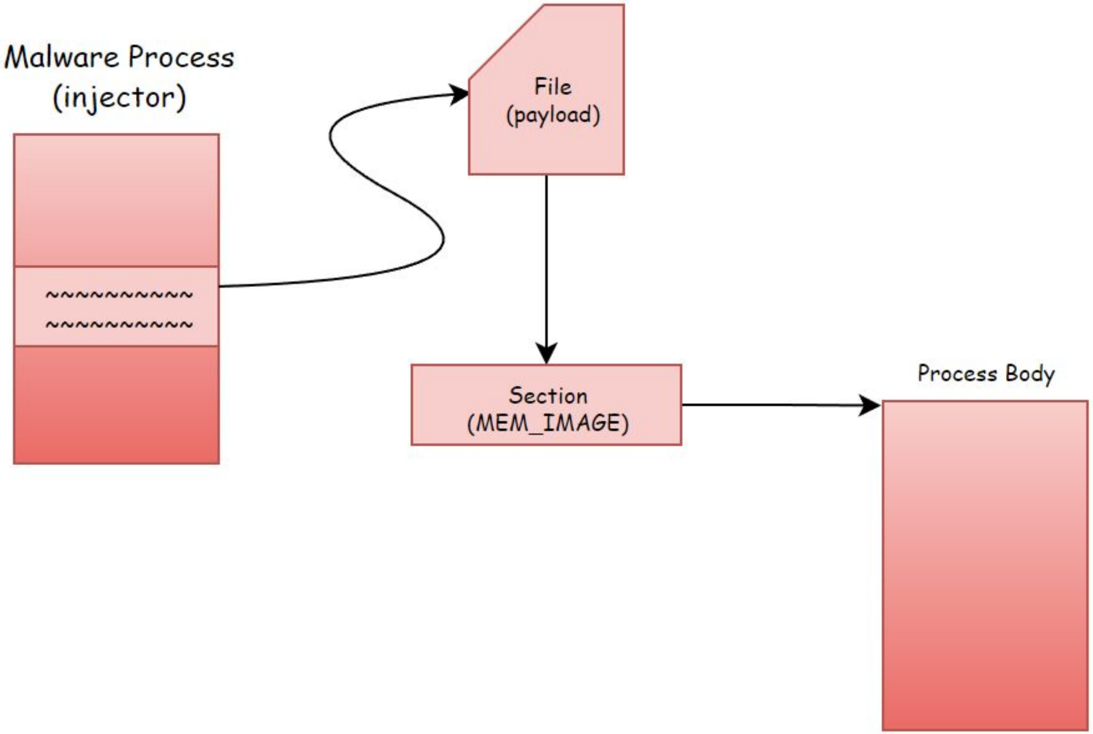
Process Herpaderping


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Write
File
(payload)

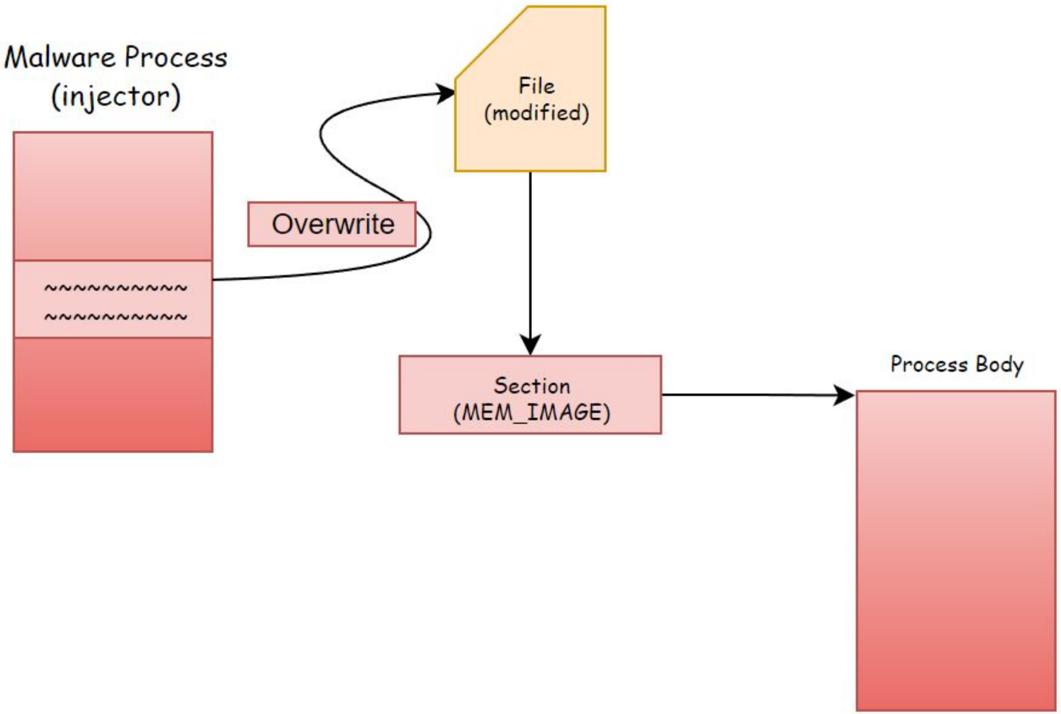
Process Herpaderping


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Write
File
(payload)
Create
Section
(MEM_IMAGE)

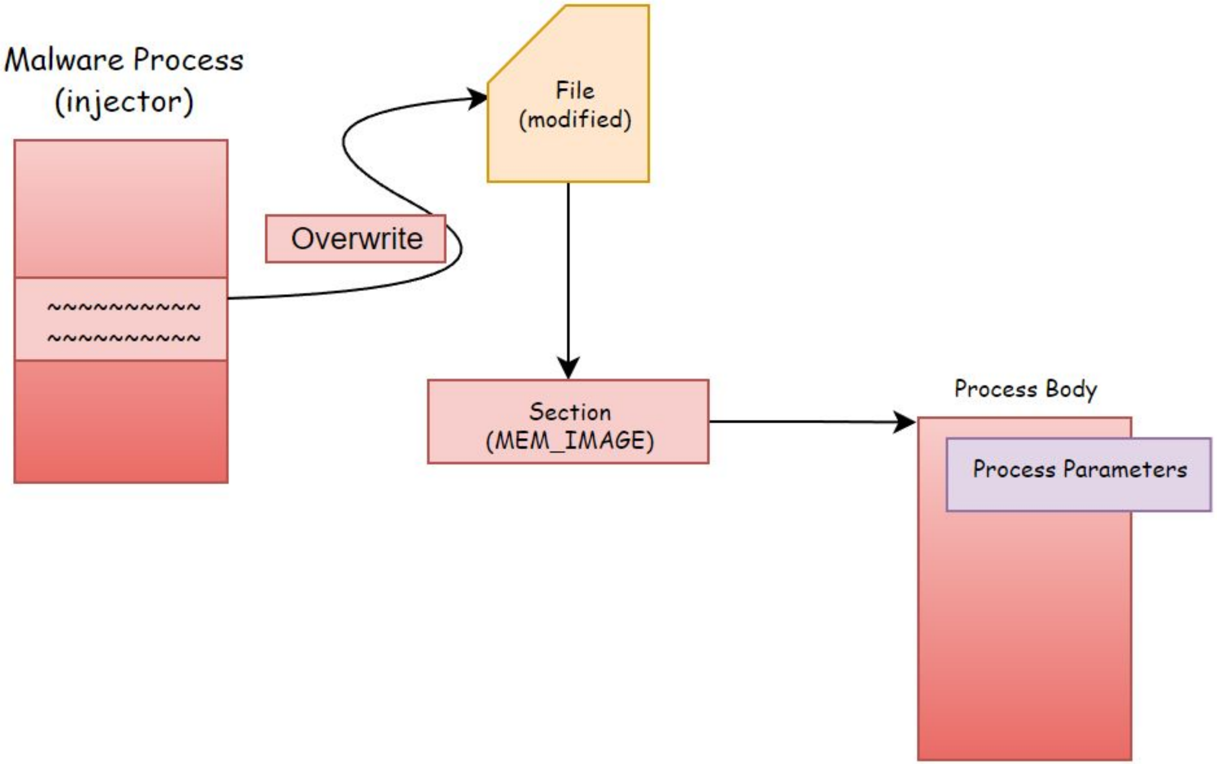
Process Herpaderping


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
File
(payload)
Section
(MEM_IMAGE)
Process Body

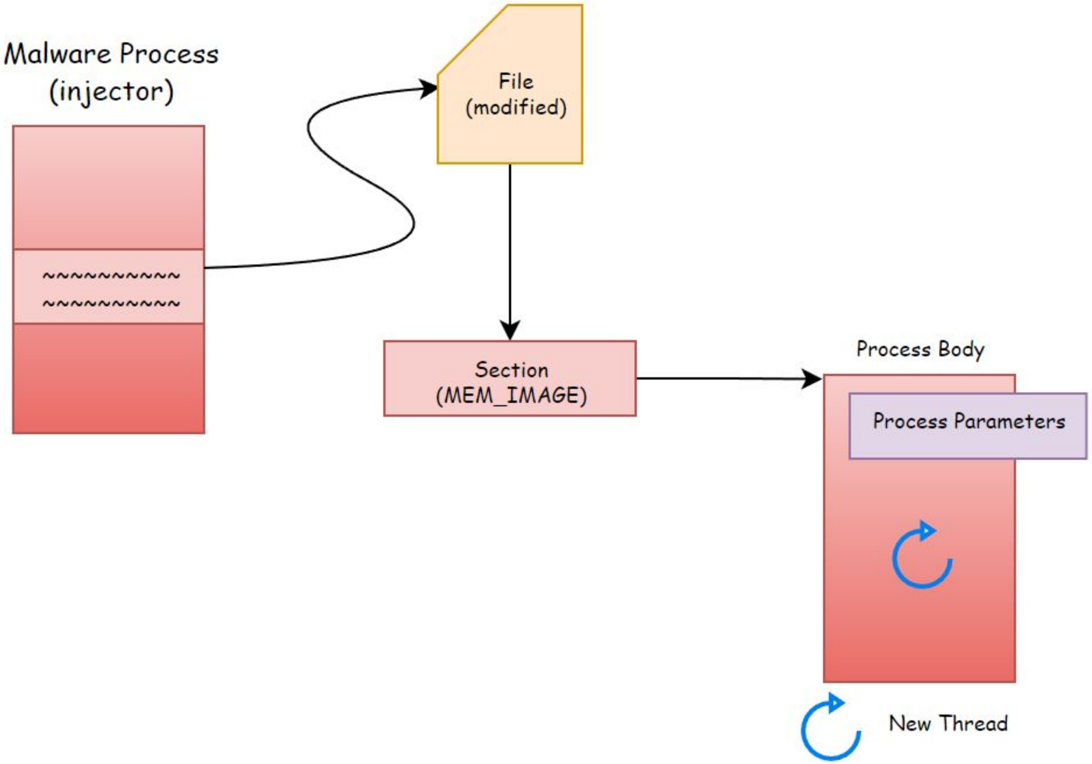
Process Herpaderping


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Overwrite
File
(modified)
Section
(MEM_IMAGE)
Process Body

Process Herpaderping


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
Overwrite
File
(modified)
Section
(MEM_IMAGE)
Process Body
Process Parameters

Process Herpaderping


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
File
(modified)
Section
(MEM_IMAGE)
Process Body
Process Parameters
New Thread

Process Herpaderping - API Calls

- Kernel32.dll:
  - CreateFileW, WriteFile, SetFilePointer,CloseHandle
- Ntdll.dll:
  - NtOpenFile, NtSetInformationFile, NtCreateSection, NtCreateProcessEx, NtCreateProcessParametersEx,NtCreateThreadEx

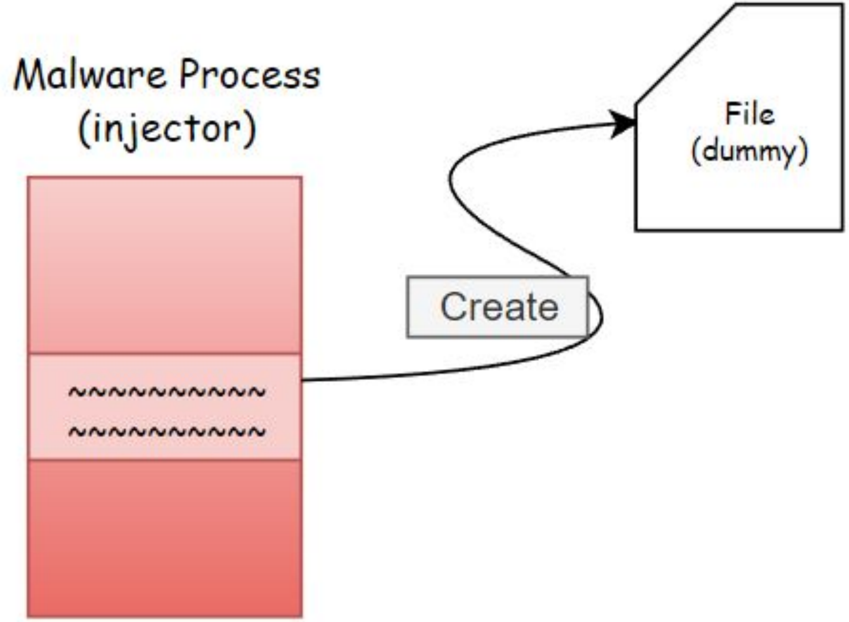
Process Ghosting

- Similar to process doppelganging
- However, the section is created using delete-pending file instead of transaction.
- Puts a file in delete-pending state which makes antivirus tools difficult to scan or delete it.
- Before creating the process the file is completely vanished and we're left out with file-less section.

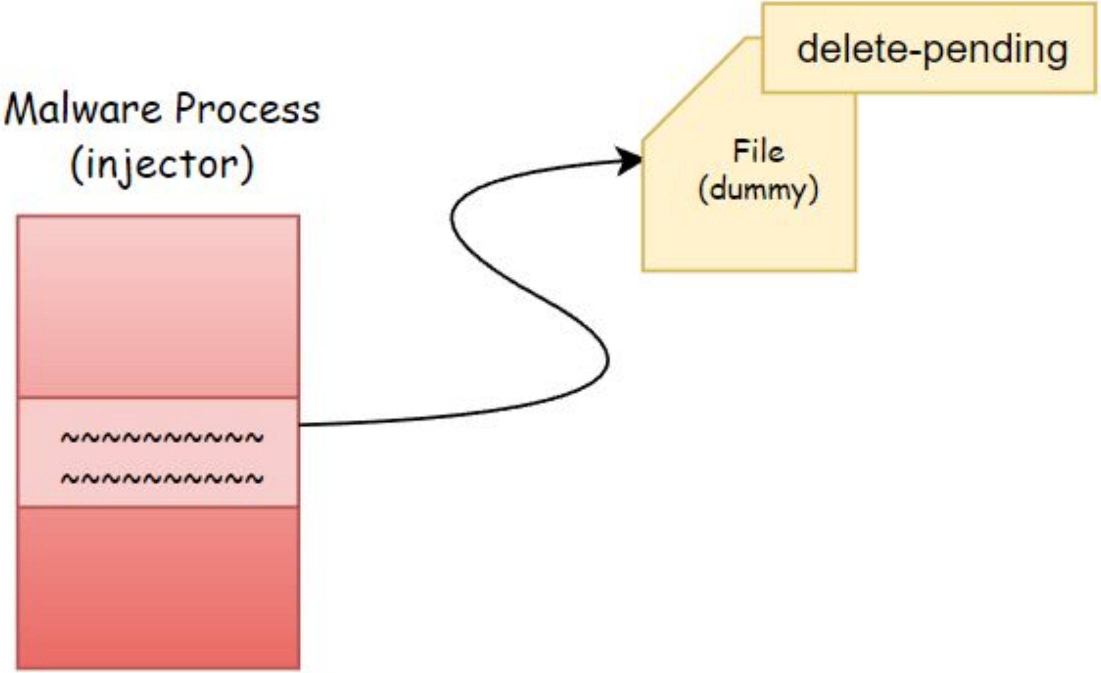
Process Ghosting - Steps

- Open/Create new dummy file
  - *CreateFileA*
- Put the file into delete-pending state using API *NtSetinformationFile*
  - ***FileDispositionInformation*** information class is used here
- Write payload buffer into delete-pending file
  - *WriteFile*
- Create an image section with the delete-pending file
  - *NtCreateSection*
- Close delete-pending file handle
  - *CloseHandle*
- Create a process with newly created image section using API *NtCreateProcessEx*
- Update/fix process parameters
  - *RtlCreateProcessParametersEx*
- Create a new thread
  - *NtCreateThreadEx*

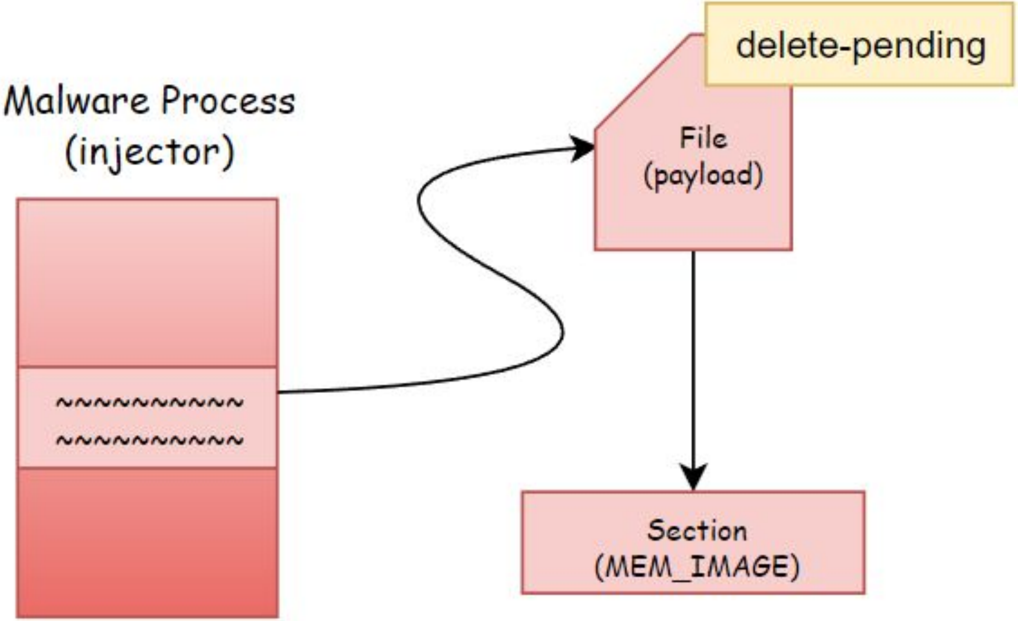
Process Ghosting


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Create
File
(dummy)

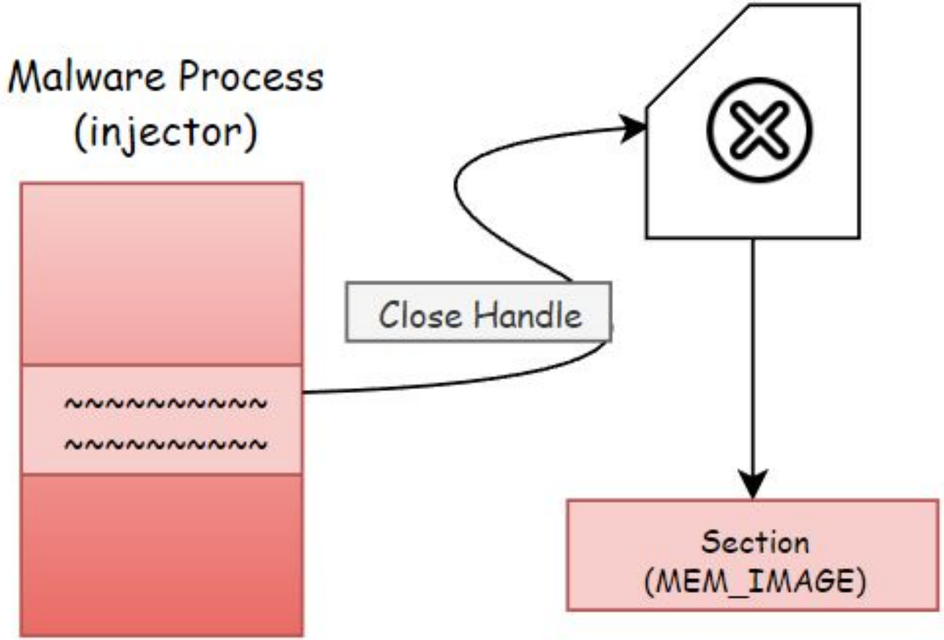
Process Ghosting


delete-pending
Malware Process
(injector)
File
(dummy)
~~~~~~~~~~
~~~~~~~~~~

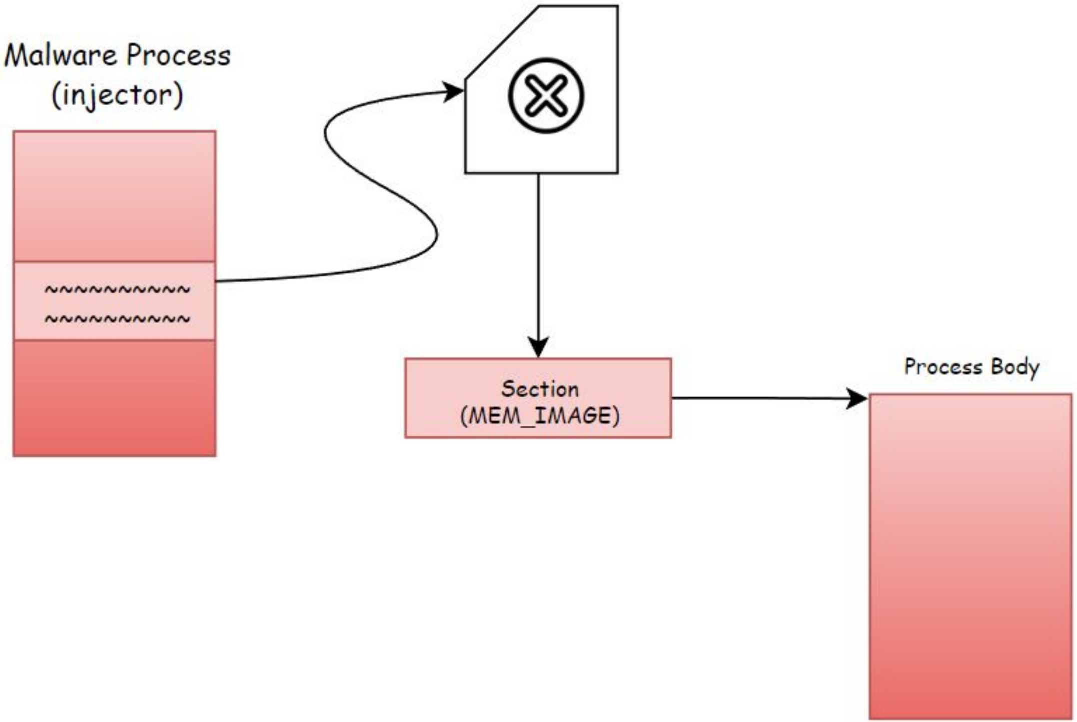
Process Ghosting


delete-pending
Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
File
(payload)
Section
(MEM_IMAGE)

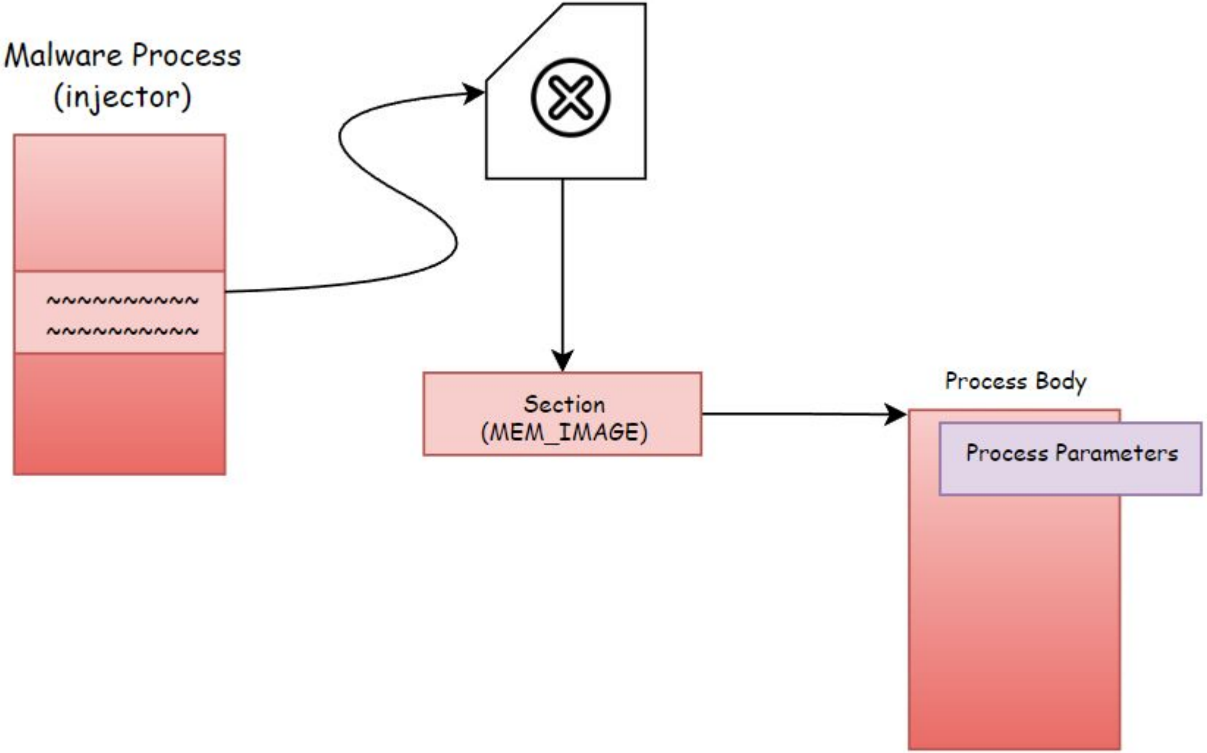
Process Ghosting


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Close Handle
Section
(MEM_IMAGE)

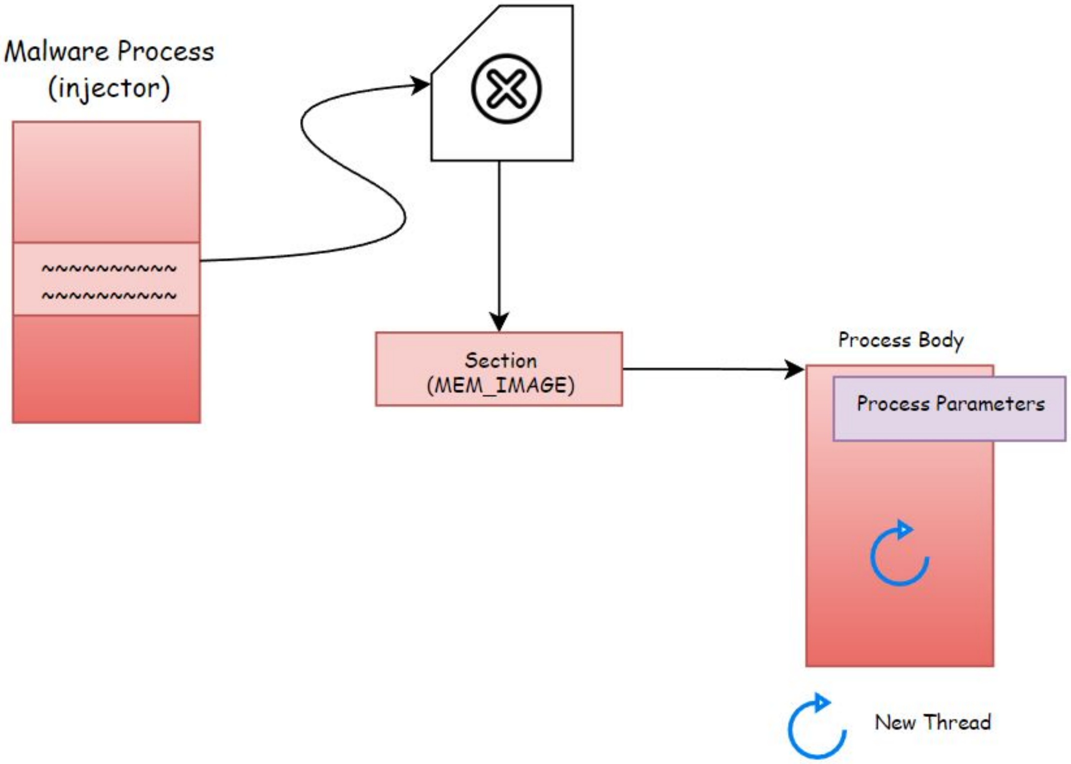
Process Ghosting


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
Section
(MEM_IMAGE)
Process Body

Process Ghosting


Malware Process
(injector)
~~~~~~~~~~~
~~~~~~~~~~~
Section
(MEM_IMAGE)
Process Body
Process Parameters

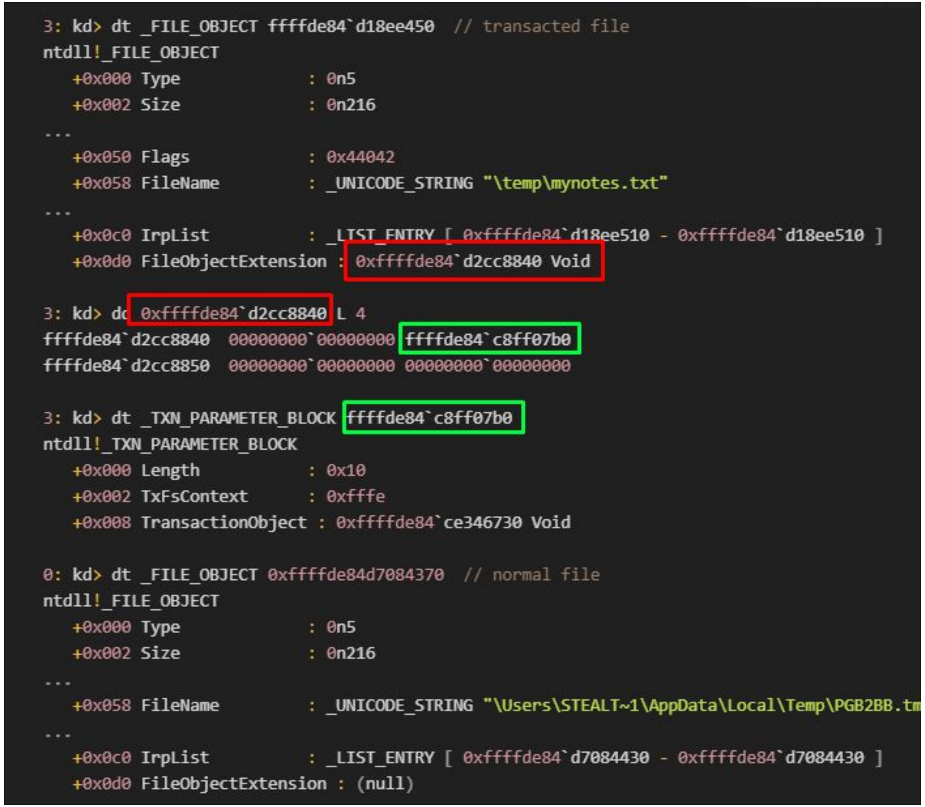
Process Ghosting


Malware Process
(injector)
~~~~~~~~~~
~~~~~~~~~~
Section
(MEM_IMAGE)
Process Body
Process Parameters
New Thread

Process Ghosting - API Calls

- Kernel32.dll:
  - WriteFile, CloseHandle
- Ntdll.dll:
  - NtOpenFile, NtSetInformationFile, NtCreateSection, NtCreateProcessEx, NtCreateProcessParametersEx,NtCreateThreadEx

Why Process Doppelganging was detected?



```
3: kd> dt _FILE_OBJECT ffffde84`d18ee450  // transacted file
ntdll!_FILE_OBJECT
   +0x000 Type             : 0n5
   +0x002 Size             : 0n216
...
   +0x050 Flags            : 0x44042
   +0x058 FileName         : _UNICODE_STRING "\temp\mynotes.txt"
...
   +0x0c0 IrpList          : _LIST_ENTRY [ 0xffffde84`d18ee510 - 0xffffde84`d18ee510 ]
   +0x0d0 FileObjectExtension : 0xffffde84`d2cc8840 Void

3: kd> dq 0xffffde84`d2cc8840 L 4
ffffde84`d2cc8840  00000000`00000000 ffffde84`c8ff07b0
ffffde84`d2cc8850  00000000`00000000 00000000`00000000

3: kd> dt _TXN_PARAMETER_BLOCK ffffde84`c8ff07b0
ntdll!_TXN_PARAMETER_BLOCK
   +0x000 Length           : 0x10
   +0x002 TxFsContext      : 0xfffe
   +0x008 TransactionObject : 0xffffde84`ce346730 Void

0: kd> dt _FILE_OBJECT 0xffffde84d7084370  // normal file
ntdll!_FILE_OBJECT
   +0x000 Type             : 0n5
   +0x002 Size             : 0n216
...
   +0x058 FileName         : _UNICODE_STRING "\Users\STEALT~1\AppData\Local\Temp\PGB2BB.tm
...
   +0x0c0 IrpList          : _LIST_ENTRY [ 0xffffde84`d7084430 - 0xffffde84`d7084430 ]
   +0x0d0 FileObjectExtension : (null)
```

References

1. https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/section-objects-and-views
2. https://www.ired.team/
3. https://blog.f-secure.com/hiding-malicious-code-with-module-stomping/
4. https://github.com/m0n0ph1/Process-Hollowing
5. https://www.youtube.com/watch?v=XmWOj-cfixs
6. https://github.com/hasherezade/transacted_hollowing
7. https://jxy-s.github.io/herpaderping/
8. https://github.com/hasherezade/process_ghosting