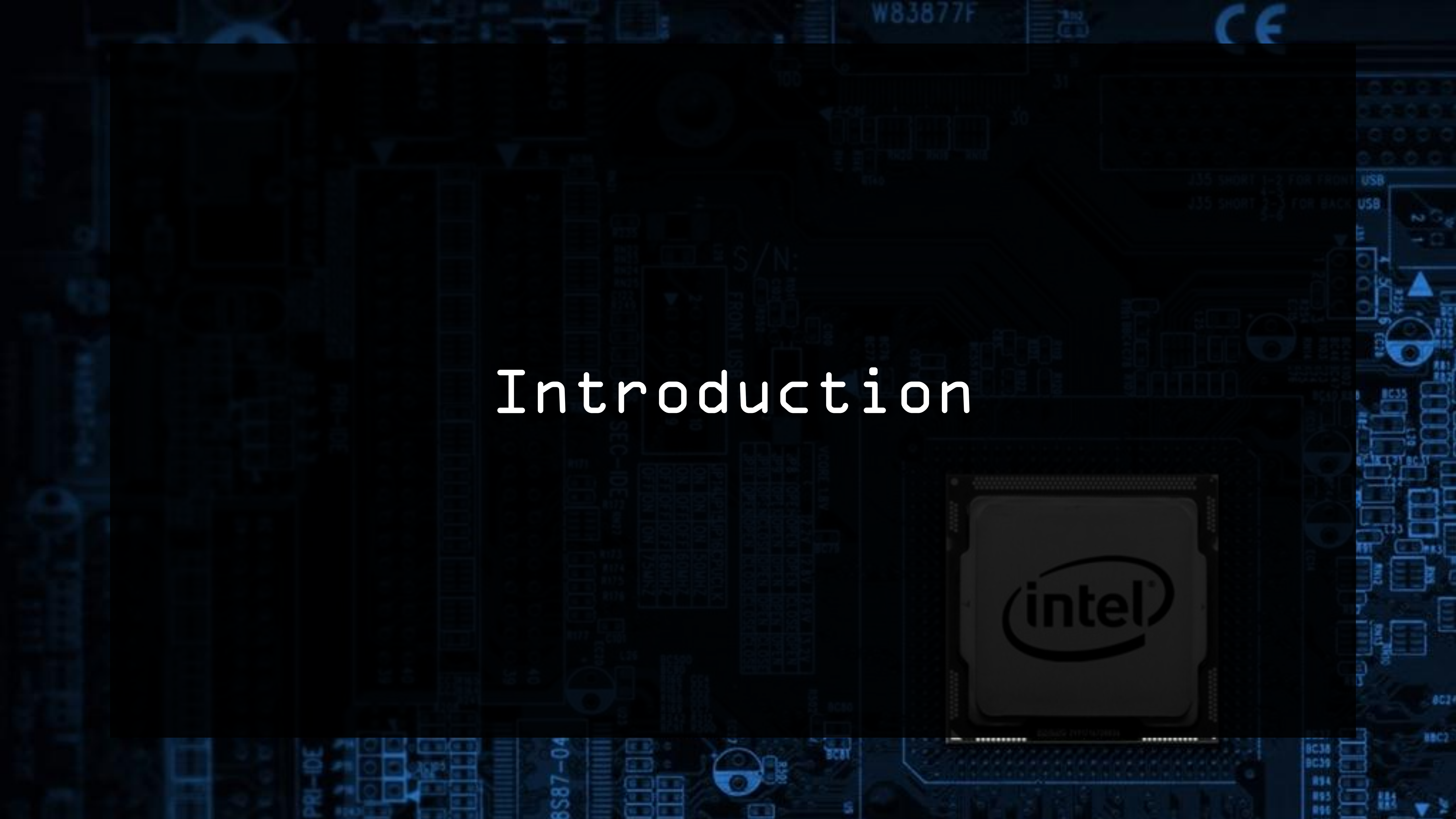
Module 1

A journey from high level languages, through assembly, to the running process

https://github.com/hasherezade/malware_training_vol1



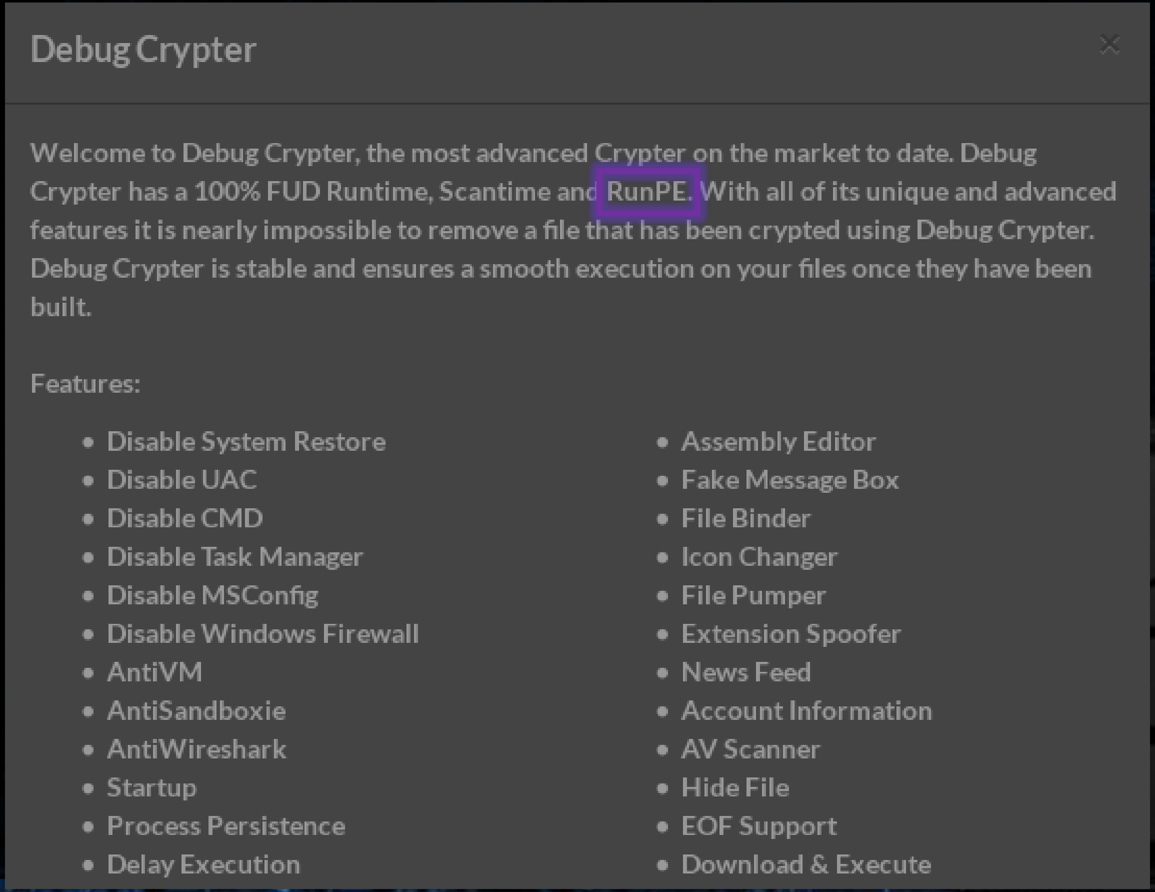
Introduction

PE injections in malware

- At various stages of execution, malware may inject its implants to other processes
  - Typical goals: process impersonation, API hooking
- Every malware author wants to avoid dropping the malicious file on the disk, so various flavors of manual loading are deployed
  - The official Win API does not support loading file from a memory buffer (only from a file)
- Almost every malware crypter uses some technique of PE injection

PE injections in malware

- Crypters



Debug Crypter

Welcome to Debug Crypter, the most advanced Crypter on the market to date. Debug Crypter has a 100% FUD Runtime, Scantime and RunPE. With all of its unique and advanced features it is nearly impossible to remove a file that has been crypted using Debug Crypter. Debug Crypter is stable and ensures a smooth execution on your files once they have been built.

Features:

- Disable System Restore
- Disable UAC
- Disable CMD
- Disable Task Manager
- Disable MSConfig
- Disable Windows Firewall
- AntiVM
- AntiSandboxie
- AntiWireshark
- Startup
- Process Persistence
- Delay Execution
- Assembly Editor
- Fake Message Box
- File Binder
- Icon Changer
- File Pumper
- Extension Spoofer
- News Feed
- Account Information
- AV Scanner
- Hide File
- EOF Support
- Download & Execute

https://blog.malwarebytes.com/threat-analysis/2015/12/malware-crypters-the-deceptive-first-layer/

Techniques of PE injection

Manual loading of EXE file

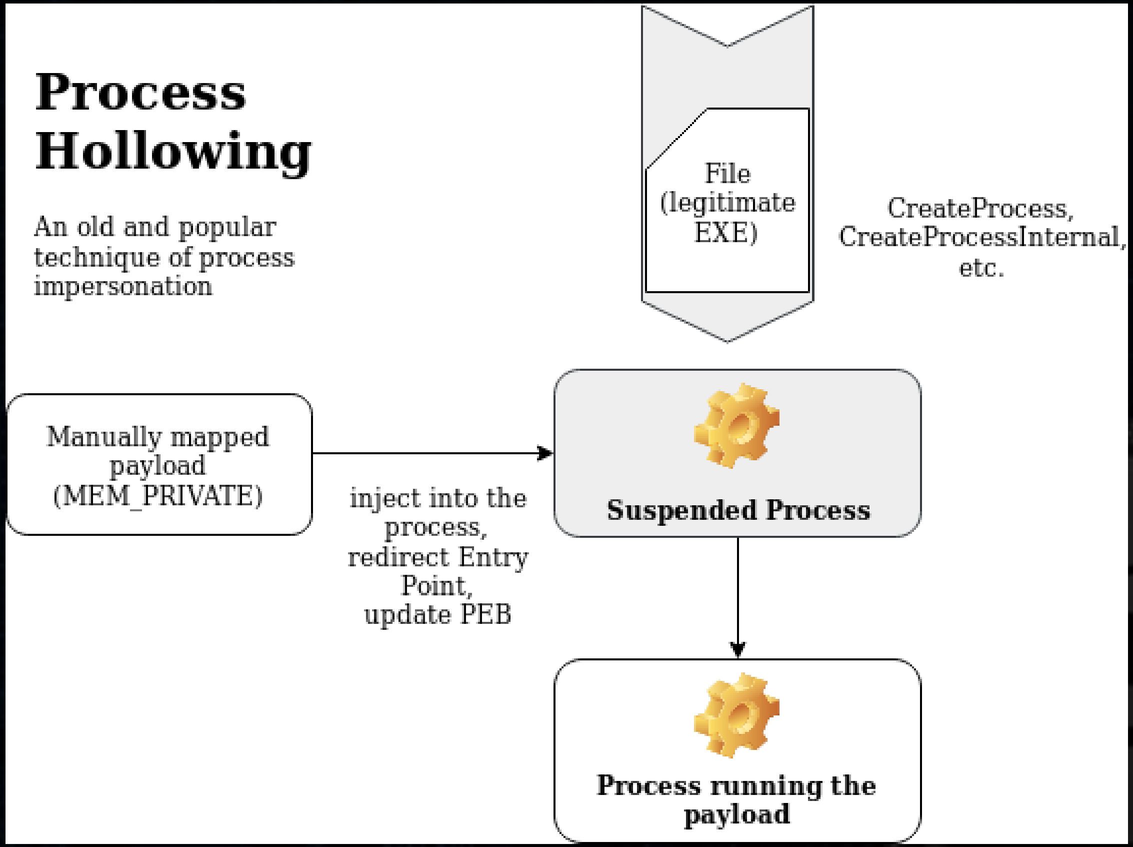
1. Map from Raw Format into Virtual Format
2. Apply relocations
3. Fill imports
4. ~~Connect to PEB~~
5. Execute the code (create a new thread of redirect execution of an existing thread)

Process Hollowing

1. Map from Raw Format into Virtual Format
2. Apply relocations
3. ~~Fill imports~~
4. Connect to PEB
5. Execute the code: redirect the Entry Point

Manual PE loading

- Process Hollowing



Process Hollowing

An old and popular technique of process impersonation

File (legitimate EXE)

CreateProcess, CreateProcessInternal, etc.

Manually mapped payload (MEM_PRIVATE)

inject into the process, redirect Entry Point, update PEB

Suspended Process

Process running the payload

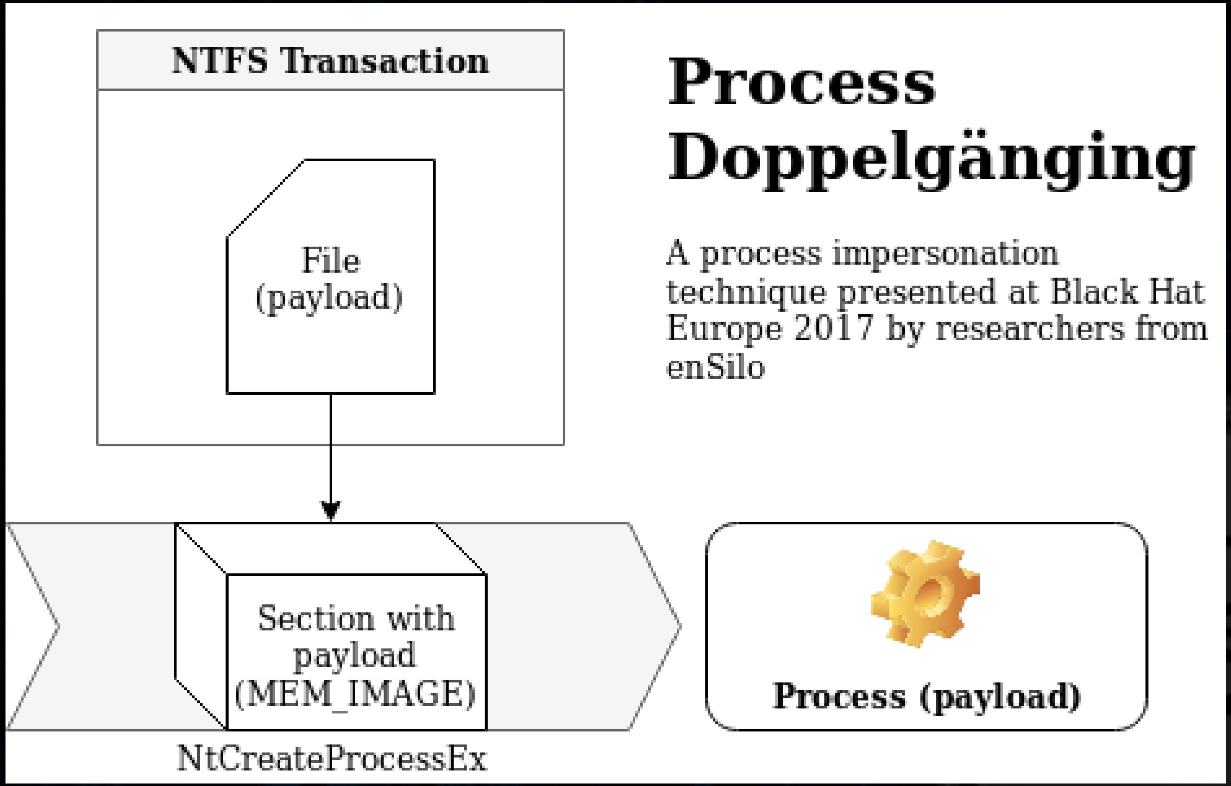
https://github.com/hasherezade/libpeconv/tree/master/run_pe

Process Doppelganging

- Map from Raw Format into Virtual Format (create a Section)
- ~~Apply relocations~~
- ~~Fill imports~~
- Execute the code: create the process out of the Section

Process Doppelganging

- Overview



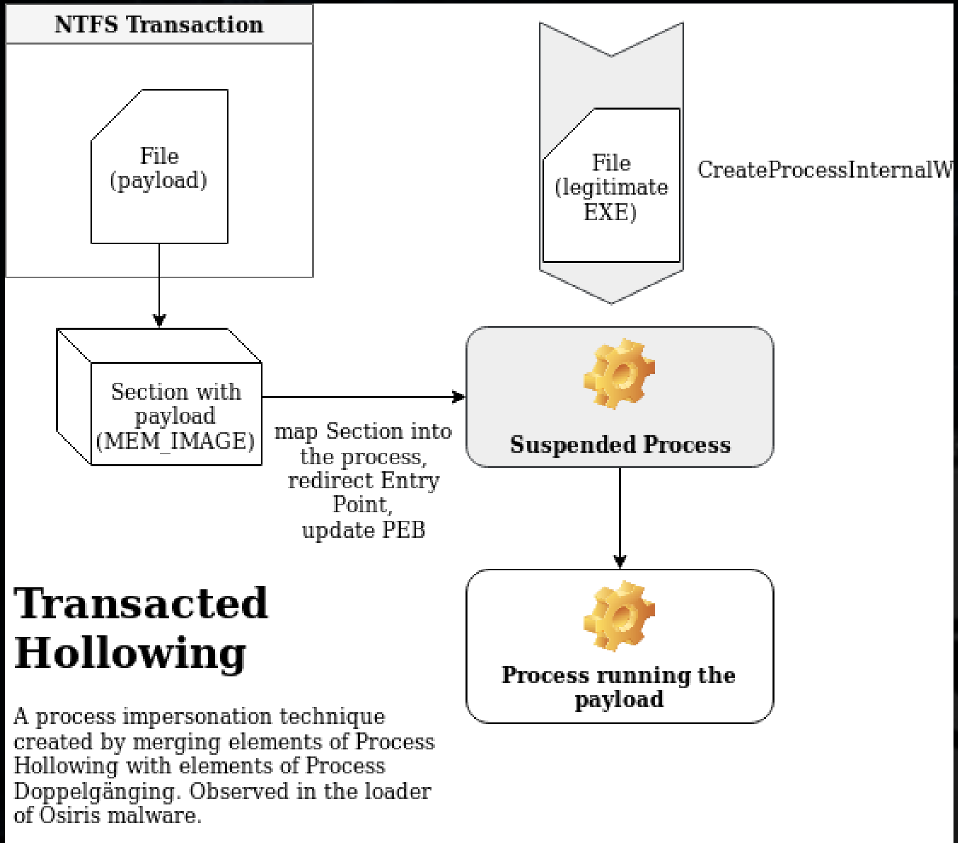
Process Doppelgänging

A process impersonation technique presented at Black Hat Europe 2017 by researchers from enSilo

https://github.com/hasherezade/process_doppelganging

Transacted Hollowing

- Overview


NTFS Transaction
File (payload)
File (legitimate EXE)
CreateProcessInternalW
Section with payload (MEM_IMAGE)
map Section into the process, redirect Entry Point, update PEB
Suspended Process
Process running the payload

Transacted Hollowing

A process impersonation technique created by merging elements of Process Hollowing with elements of Process Doppelgänging. Observed in the loader of Osiris malware.

Module Overloading

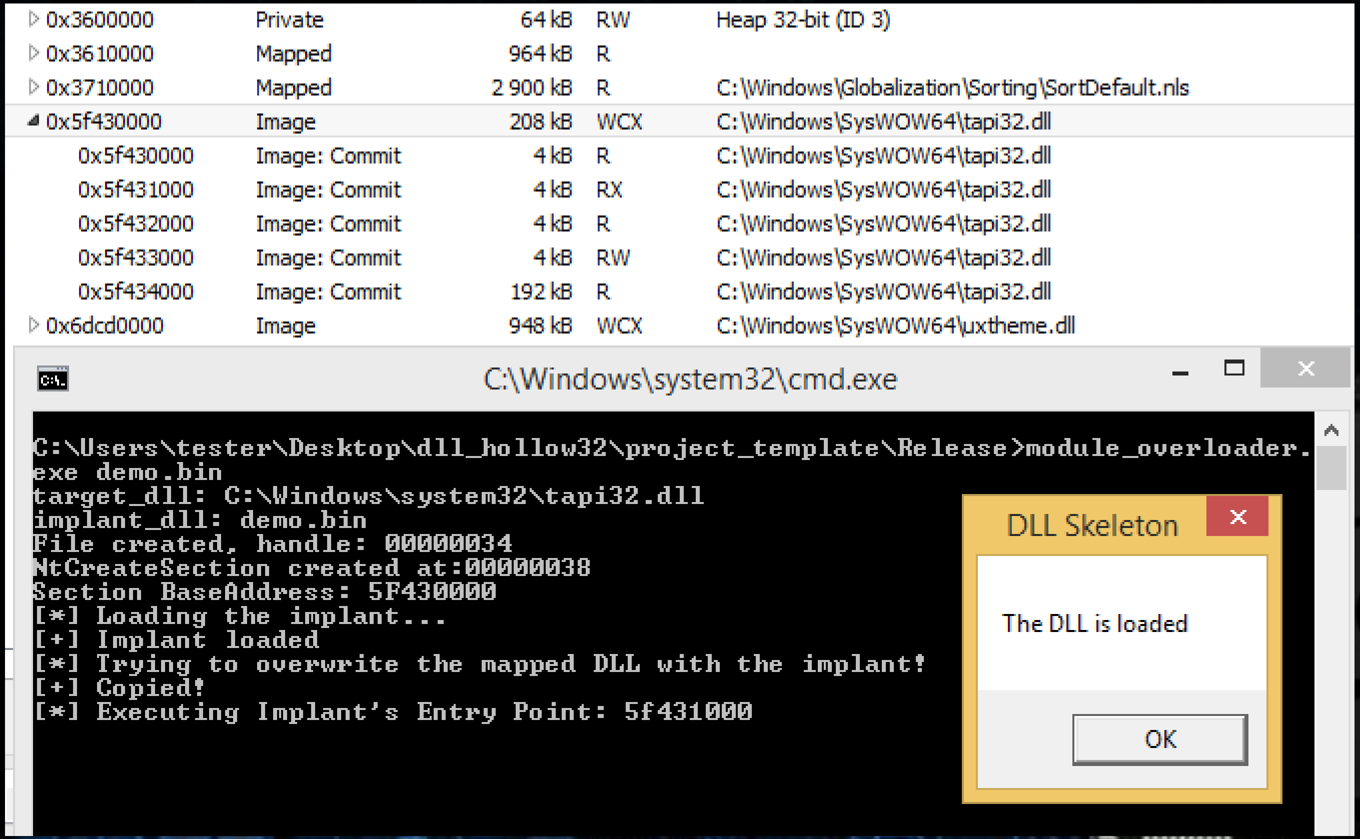
- An idea of @TheRealWover
- PoC implemented by me
  - https://github.com/hasherezade/module_overloading
- Similar to DLL hollowing, but the implant is not connected to the list of modules (may deceive some tools that search for the artefacts typical for hollowing)

Module Overloading

1. Load a target DLL as MEM_IMAGE
2. Load the implant DLL manually (with filling imports)
3. Relocate the implant to the target base
4. Overwrite the target image with the implant
5. Fetch implant's Entry Point
6. Execute the implant

Module Overloading

In action:



▷ 0x3600000	Private	64 kB	RW	Heap 32-bit (ID 3)
▷ 0x3610000	Mapped	964 kB	R	
▷ 0x3710000	Mapped	2 900 kB	R	C:\Windows\Globalization\Sorting\SortDefault.nls
◢ 0x5f430000	Image	208 kB	WCX	C:\Windows\SysWOW64\tapi32.dll
0x5f430000	Image: Commit	4 kB	R	C:\Windows\SysWOW64\tapi32.dll
0x5f431000	Image: Commit	4 kB	RX	C:\Windows\SysWOW64\tapi32.dll
0x5f432000	Image: Commit	4 kB	R	C:\Windows\SysWOW64\tapi32.dll
0x5f433000	Image: Commit	4 kB	RW	C:\Windows\SysWOW64\tapi32.dll
0x5f434000	Image: Commit	192 kB	R	C:\Windows\SysWOW64\tapi32.dll
▷ 0x6dcd0000	Image	948 kB	WCX	C:\Windows\SysWOW64\uxtheme.dll

```
C:\Users\tester\Desktop\dll_hollow32\project_template\Release>module_overloader.
exe demo.bin
target_dll: C:\Windows\system32\tapi32.dll
implant_dll: demo.bin
File created, handle: 00000034
NtCreateSection created at:00000038
Section BaseAddress: 5F430000
[*] Loading the implant...
[+] Implant loaded
[*] Trying to overwrite the mapped DLL with the implant!
[+] Copied!
[*] Executing Implant's Entry Point: 5f431000
```

DLL Skeleton

The DLL is loaded

https://github.com/hasherezade/module_overloading

Exercise 1

- Let's take a look at the implementation
  - Process Hollowing (aka Run PE):
    - https://github.com/hasherezade/libpeconv/blob/master/run_pe
  - Process Doppelganging:
    - https://github.com/hasherezade/process_doppelganging
  - Module Overloading:
    - https://github.com/hasherezade/module_overloading