BUILDING YOUR FIRST WINDOWS MALWARE LOADER

ROYCE YAEZENKO

0xtengu

https://github.com/0xtengu/rtv_workshop

AGENDA

1. ORIENTATION
2. WINDOWS INTERNALS LITE
3. PE LITE
4. PROCESS INJECTION 101
5. WHAT'S NEXT?

ENVIRONMENT CHECKLIST

WINDOWS

- ❏ Windows 11
- ❏ Snapshots
- ❏ Visual Studio 2022

LINUX

- ❏ Shell
- ❏ msfconsole

OPTIONAL

- ❏ PE-bear / DIE
- ❏ X96dbg
- ❏ System Informer / Process Hacker 2

EXPECTATIONS

- ❑ WHAT YOU'LL GET:
  - ❑ Core Concepts
  - ❑ A Foundation
- ❑ WHAT YOU WON'T GET:
  - ❑ Ready-to-Use
  - ❑ FUD
- ❑ BY THE END, YOU'LL HAVE:
  - ❑ The ability to continue on your own

YOU DON'T NEED TO REINVENT THE WHEEL

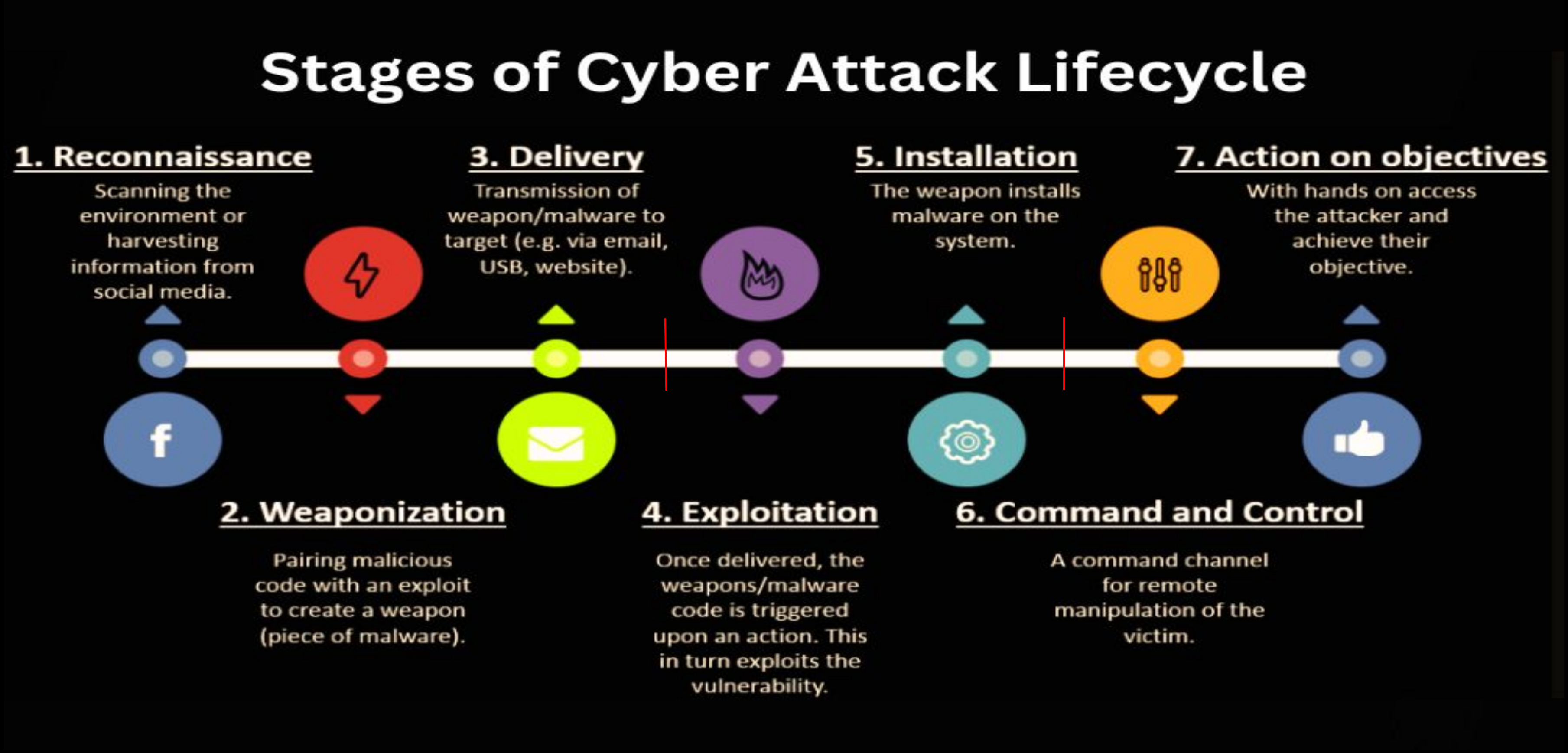
DISCLAIMER

WINDOWS MALWARE DEVELOPMENT
WHAT AND WHY

- ❏ OXFORD LANGUAGE:
  - ❏ Software that is specifically designed to disrupt, damage, or gain unauthorized access to a computer system
  - ❏ Malicious Software
- ❏ WHAT LANGUAGE?

- ❏ WHY WINDOWS?
  - ❏ Enterprise Dominance
- ❏ WHY BUILD IT?
  - ❏ Help Defenders
- ❏ LOADER VS PAYLOAD:
  - ❏ Loader = Trojan Horse
  - ❏ Payload = Soldiers Inside

LET'S SET THE SCENE



Stages of Cyber Attack Lifecycle

1. Reconnaissance
Scanning the environment or harvesting information from social media.

2. Weaponization
Pairing malicious code with an exploit to create a weapon (piece of malware).

3. Delivery
Transmission of weapon/malware to target (e.g. via email, USB, website).

4. Exploitation
Once delivered, the weapons/malware code is triggered upon an action. This in turn exploits the vulnerability.

5. Installation
The weapon installs malware on the system.

6. Command and Control
A command channel for remote manipulation of the victim.

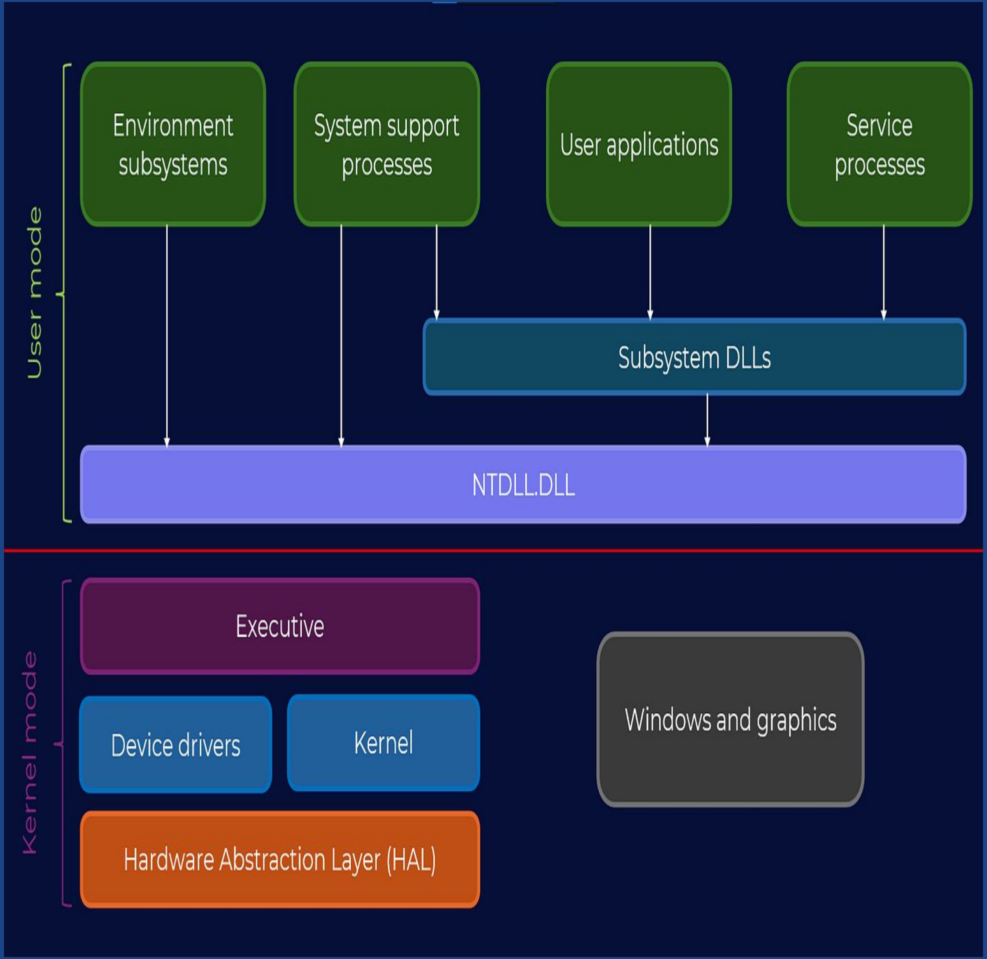
7. Action on objectives
With hands on access the attacker and achieve their objective.

WINDOWS

INTERNALS LITE

VOL. 1: ARCHITECTURE

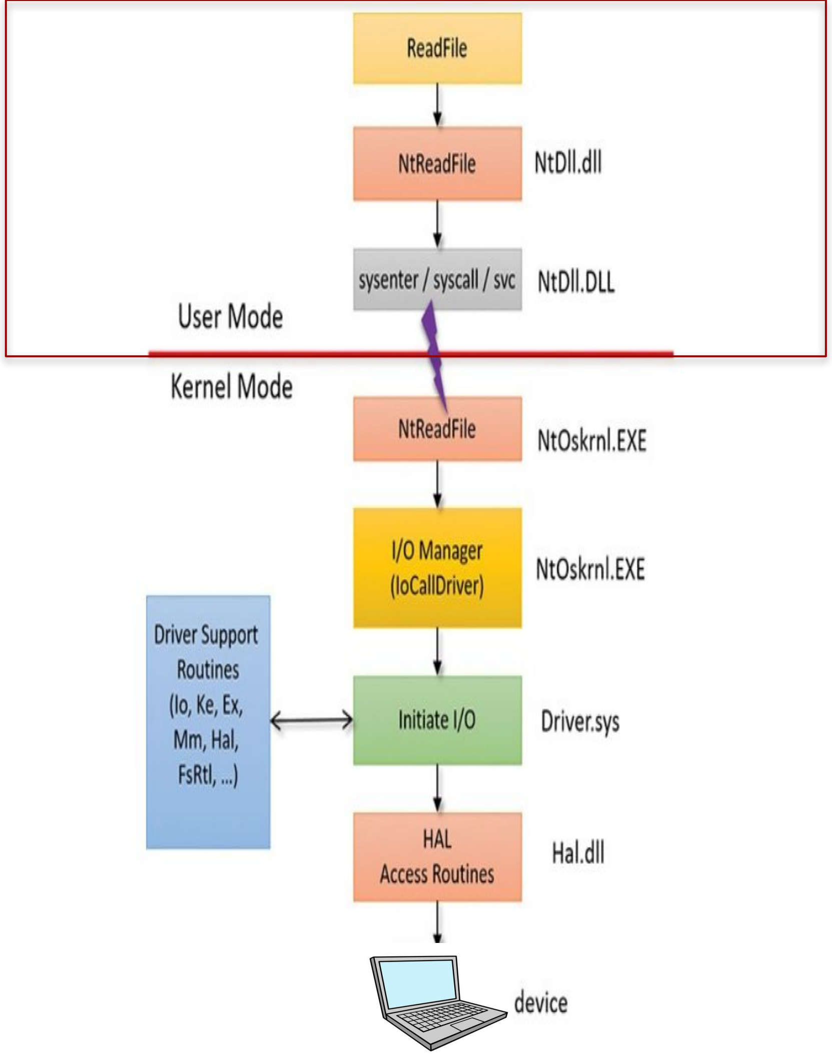
- USER APPLICATIONS
  - Notepad, Calculator
- SUBSYSTEM DLLs (WINAPI)
  - Kernel32.dll, user32.dll, etc.
  - Provides standard API funcs
- NATIVE API LAYER
  - Ntdll.dll
  - Implements Nt* funcs
  - Lowester userland dll
- KERNEL
  - C:\Windows\System32
    - ntoskrnel.exe



ARCHITECTURE CONTINUED

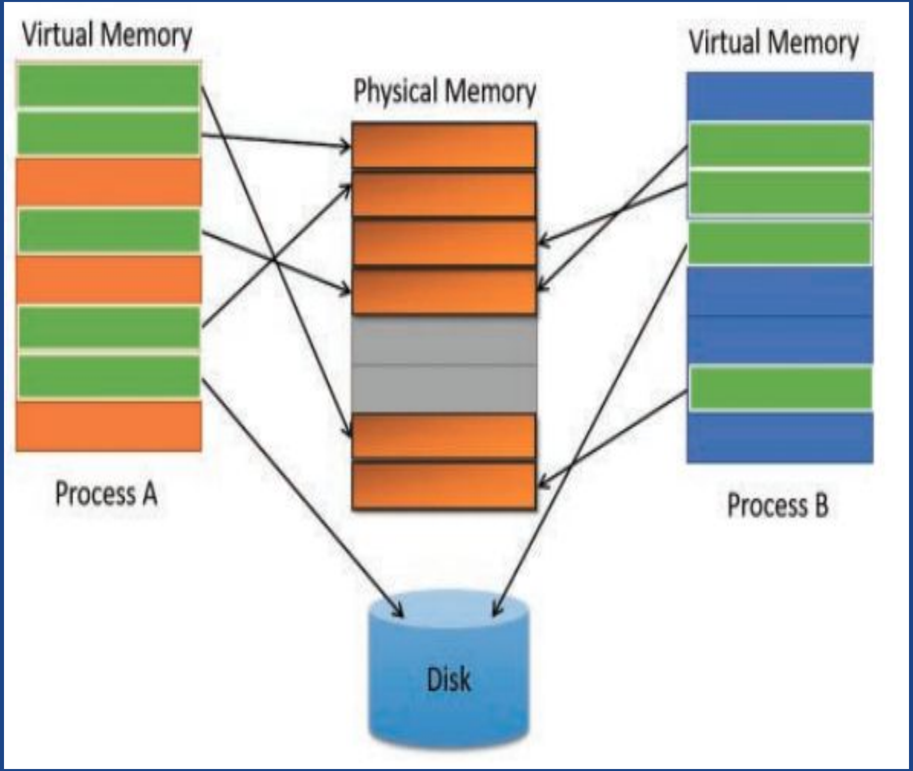
- ❑ App calls from ReadFile()
  - ❑ From kernel32.dll (WinAPI)
- ❑ Kernel32.dll calls NtReadFile()
  - ❑ From ntdll.dll
- ❑ Ntdll.dll performs system call
  - ❑ syscall(x86) or sysenter(x64)
- ❑ Windows Kernel handles:
  - ❑ I/O
  - ❑ drivers
  - ❑ etc.

SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE
Hunt Evil
POSTER



VOL. 2: MEMORY MANAGEMENT

- MEMORY SPACE
  - X86 = 0XFFFFFFFF
  - X64 = 0XFFFFFFFFFFFFFFFF
- EACH PROCESS
  - Individual virtual address space
- 4KB PAGES
  - Page tables map virtual -> physical
  - Free, Reserved, Committed



MEMORY CONTINUED

- PROTECTION FLAGS
  - PAGE_EXECUTE_READWRITE - RWX
  - There are more variations
- VirtualAllocEx
  - MEM_RESERVE | MEM_COMMIT -> both at once
- VirtualProtectEx
  - Lets us tighten perms after copying our shellcode (RW -> RX)

EXAMPLE

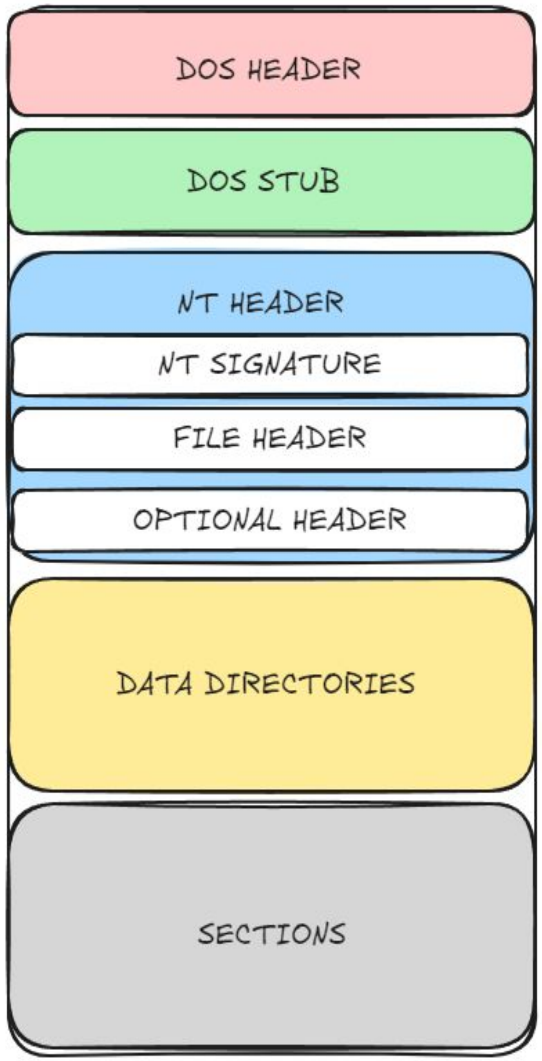
VOL. 3: WINDOWS API (WINAPI)

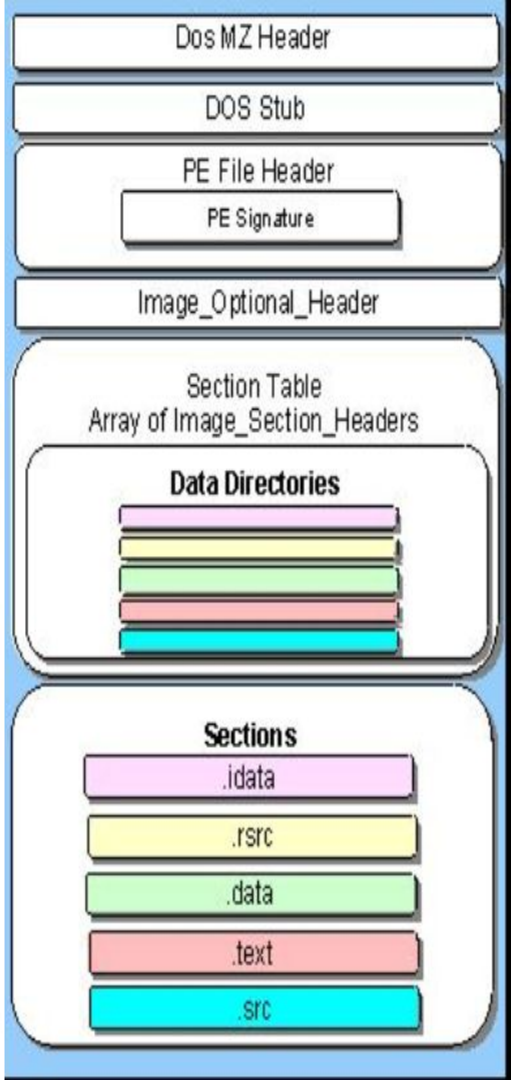
- WINDOWS APPLICATION PROGRAMMING INTERFACE
  - The core functions, interfaces, and libraries that allow applications to interact with the Windows Operating System
- HAS VARIOUS DATA TYPES
  - https://learn.microsoft.com/en-us/windows/win32/winprog/windows-data-types
  - Has various types pointers: PDWORD = DWORD*
- ANSI vs. UNICODE
  - CreateFileA vs. CreateFileW
  - CreateFileA = LPCSTR vs CreateFileW = LPCWSTR

EXAMPLES

HELLO PE: BRIEFLY

- .exe .dll .sys
- DOS HEADER
  - MZ = 0x4D and 0x5A
- DOS STUB
- NT HEADER
  - IMAGE_NT_HEADERS
  - File Header
  - Optional Header
- DATA DIRECTORIES
  - Export Directory
  - Funcs and vars exported
  - IAT - Import Address Table
  - Other PE funcs addresses
- SECTION





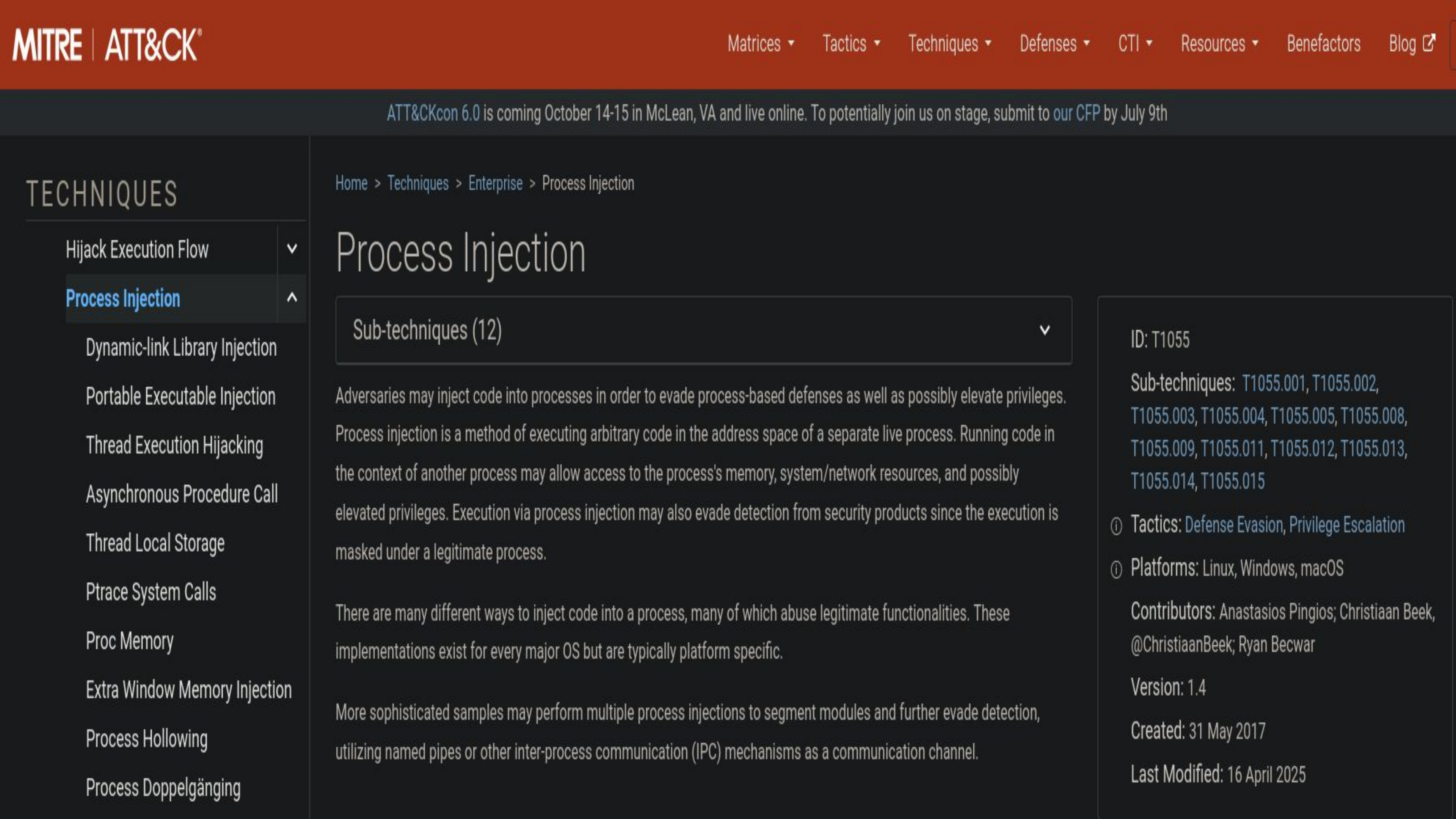
PROCESS INJECTION
(OUR LOADER)

PROCESS INJECTION 101

- DROP CODE INTO ANOTHER RUNNING PROCESS
    - ALLOWS CODE TO EXECUTE UNDER PERMISSIONS OF TARGET PROCESS
    - FORCES THE TARGET PROCESS TO RUN ARBITRARY CODE
    - INHERITS ALL ACCESS RIGHTS AND TRUST LEVEL

- PROCESS HANDLE ACQUISITION, MEMORY ALLOCATION IN TARGET, CODE WRITING OPS, EXECUTION TRIGGER

- STABILITY?



ATT&CKcon 6.0 is coming October 14-15 in McLean, VA and live online. To potentially join us on stage, submit to our CFP by July 9th

TECHNIQUES
Hijack Execution Flow
Process Injection
Dynamic-link Library Injection
Portable Executable Injection
Thread Execution Hijacking
Asynchronous Procedure Call
Thread Local Storage
Ptrace System Calls
Proc Memory
Extra Window Memory Injection
Process Hollowing
Process Doppelgänging

Home > Techniques > Enterprise > Process Injection

Process Injection

Sub-techniques (12)

Adversaries may inject code into processes in order to evade process-based defenses as well as possibly elevate privileges. Process injection is a method of executing arbitrary code in the address space of a separate live process. Running code in the context of another process may allow access to the process's memory, system/network resources, and possibly elevated privileges. Execution via process injection may also evade detection from security products since the execution is masked under a legitimate process.

There are many different ways to inject code into a process, many of which abuse legitimate functionalities. These implementations exist for every major OS but are typically platform specific.

More sophisticated samples may perform multiple process injections to segment modules and further evade detection, utilizing named pipes or other inter-process communication (IPC) mechanisms as a communication channel.

ID: T1055

Sub-techniques: T1055.001, T1055.002, T1055.003, T1055.004, T1055.005, T1055.008, T1055.009, T1055.011, T1055.012, T1055.013, T1055.014, T1055.015

Tactics: Defense Evasion, Privilege Escalation

Platforms: Linux, Windows, macOS

Contributors: Anastasios Pingios; Christiaan Beek, @ChristiaanBeek; Ryan Becwar

Version: 1.4

Created: 31 May 2017

Last Modified: 16 April 2025

WORKSHOP

OTHER TECHNIQUES

CLASSIC REMOTE INJECTION

- ❏ Remote Thread ← Our Loader
- ❏ Thread Hijacking
  - ❏ Set thread context
  - ❏ APC injection ← v2
    - ❏ Early Bird Injection

SYSCALL AND NATIVE API ABUSE

- ❏ Direct syscalls ← v2
- ❏ Indirect syscalls
  - ❏ hell’s/tartarus gate
  - ❏ syswhispers2/3

MAPPING

- ❏ Manual PE mapping
- ❏ Manual memory mapping
  - ❏ Memory section injection

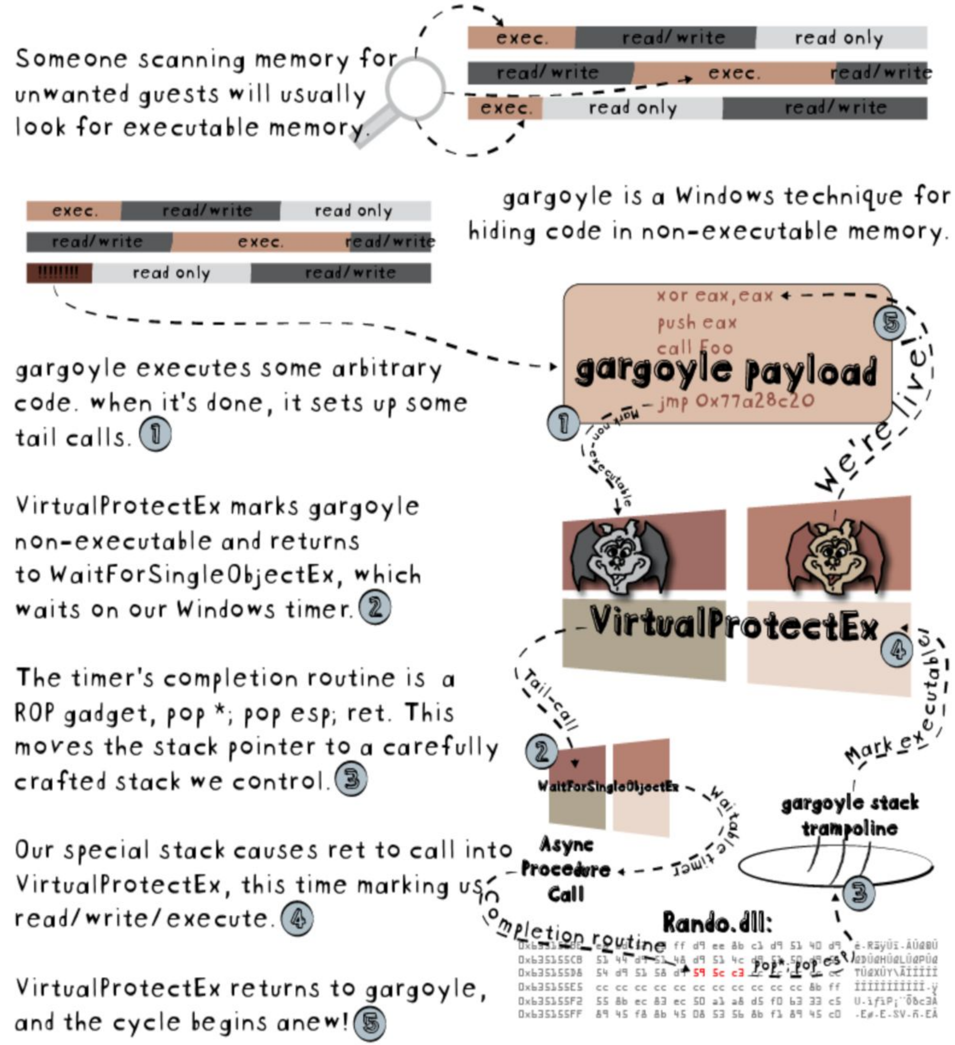
HOLLOWING

- ❏ Process Hollowing
- ❏ Process no-hollowing
- ❏ Ghostly hollowing

https://lospi.net/security/assembly/c/cpp/developing/software/2017/03/04/gargoyle-memory-analysis-evasion.html

gargoyle

a memory scanning evasion technique



Someone scanning memory for unwanted guests will usually look for executable memory.

gargoyle is a Windows technique for hiding code in non-executable memory.

gargoyle executes some arbitrary code. when it's done, it sets up some tail calls. 1

VirtualProtectEx marks gargoyle non-executable and returns to WaitForSingleObjectEx, which waits on our Windows timer. 2

The timer's completion routine is a ROP gadget, pop *; pop esp; ret. This moves the stack pointer to a carefully crafted stack we control. 3

Our special stack causes ret to call into VirtualProtectEx, this time marking us read/write/execute. 4

VirtualProtectEx returns to gargoyle, and the cycle begins anew! 5

```
xor eax,eax
push eax
call Foo
jmp 0x77a28c20
```

Rando.dll:

```
0xb35155BE  [illegible]  ff d9 ee 8b c1 d9 51 40 d9  é·RãyÜï·ÄÜ@ßÜ
0xb35155CB  51 44 d9 51 48 d9 51 4c d9 51 50 d9 51  @DÜ@HÜ@LÜ@PÜ@
0xb35155D8  54 d9 51 58 d9 59 5c c3 cc cc cc cc cc  TÜ@XÜY\ÃÌÌÌÌÌ
0xb35155E5  cc cc cc cc cc cc cc cc cc cc cc 8b ff  ÌÌÌÌÌÌÌÌÌÌÌ·ÿ
0xb35155F2  55 8b ec 83 ec 50 a1 a8 d5 f0 b3 33 c5  U·ìƒìP¡¨Õðc3Å
0xb35155FF  89 45 f8 8b 45 08 53 56 8b f1 89 45 c0  ·Eø·E·SV·ñ·EÀ
```

RESOURCES, CITATIONS, AND MORE IS ON THE GITHUB:

https://github.com/0xtengu/rtv_workshop

THANK YOU

— END OF MATERIAL —