Evasion Adventures

A brief summary of modern offensive tradecraft

ethan@kali # whoami

Currently studying infosec in SP

Interning at CSA Attack Simulation Group

ethan@kali # whoami

```
def load_code(buffer):
    allocated = VirtualAlloc(0, len(buffer), 0x3000, 0x40)
    RtlMoveMemory(allocated, buffer, len(buffer))
    handle = CreateThread(0, 0, allocated, 0, 0, 0)
    return handle


handle = load_code(shellcode)
WaitForSingleObject(handle, -1)
```

Disclaimer I am obligated to put here

I am not responsible for what you do with this information, but it would be nice if you didn’t do anything illegal with it :)

All of this information is publicly available, no 0days here

Another disclaimer I am obligated to put here

Whatever I say here is based on my personal research or from others' experiences. They do not represent the views of my school/employer

Table of contents

- Evasion in red teaming
- Traditional vs modern evasion strategy
- Explaining modern detection techniques
- Explaining modern evasion techniques
- Some demos (yay!)

Red Teaming

Purpose of a red team

- Evaluate security measures effectiveness against an advanced adversary
- Give defenders and incident responders realistic practice
- **Be the “bad” guy**

Importance of evasion in red teaming

Once detected:

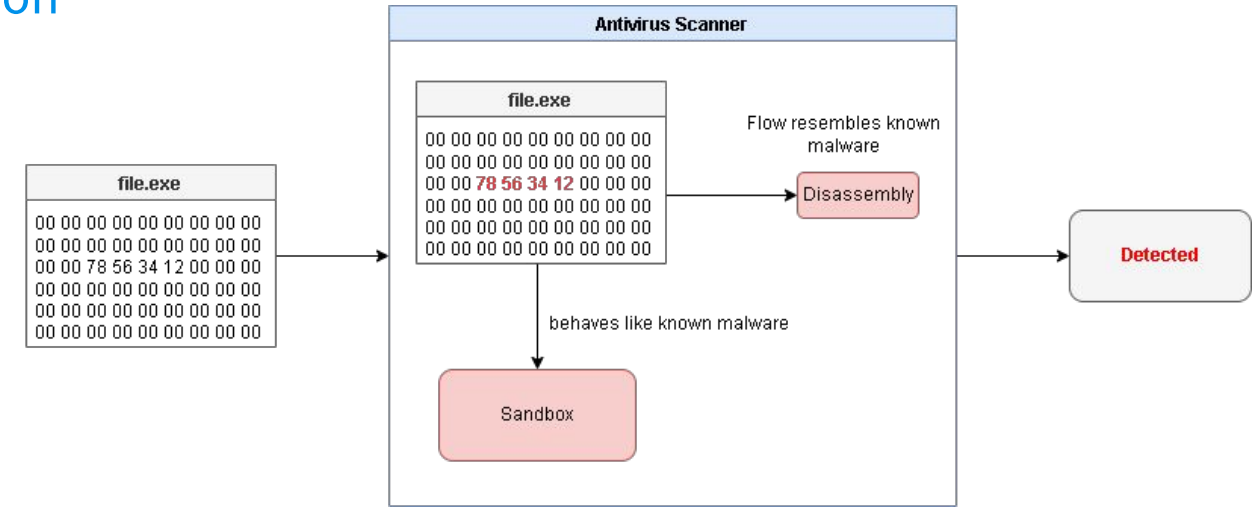
- Red team is heavily time restricted
- Defenses will be on high alert instantly
- Red team resources start to get burnt

Much harder to reach objective

Understanding the enemy

Traditional detection

Detection



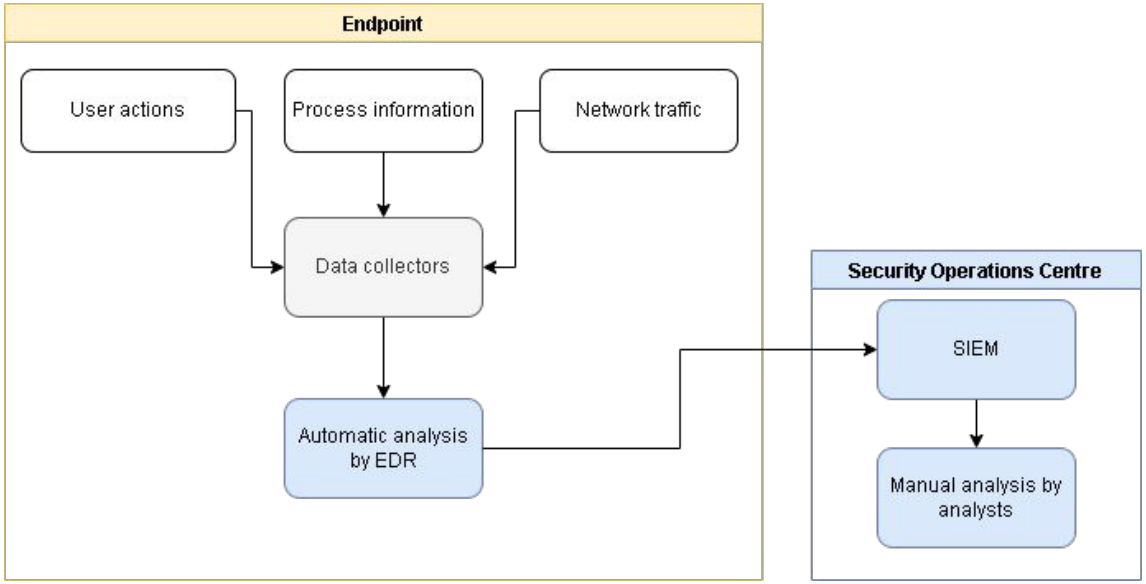
- Heavily signature based
- Sandbox analysis (heuristics)
- Detects known bad artifacts e.g. file signatures, functions etc.

Evading traditional detection

Evasion

- Obfuscation
- Anti debugging, anti disassembly, anti sandbox
- Hide **known** bad artifacts

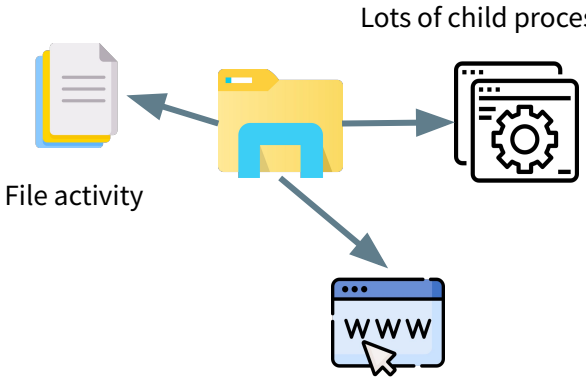
Modern detection



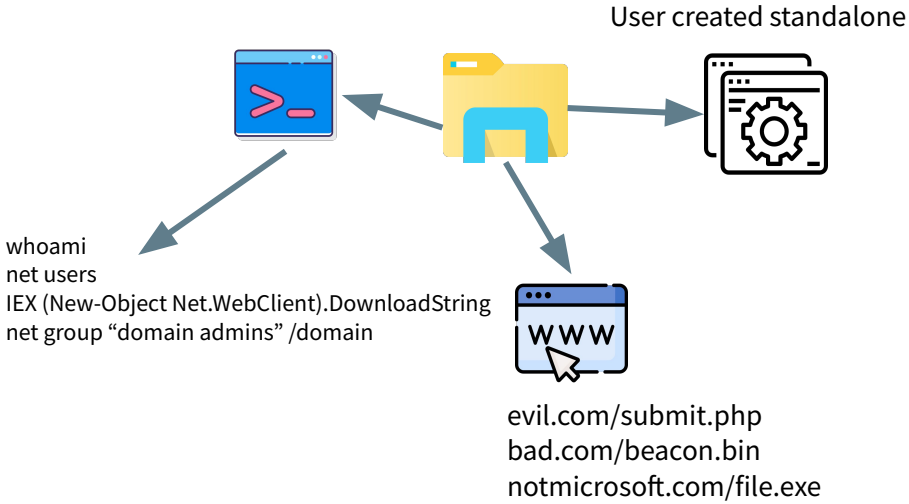
- Normal baseline vs abnormal
- Collect large amounts of telemetry data
- Flag abnormal behaviour for analysis

Modern detection

Baseline
normal - Ok



Abnormal - investigate



Modern detection

- Our payload could have evaded traditional detections
- Our payload + actions can still get us flagged

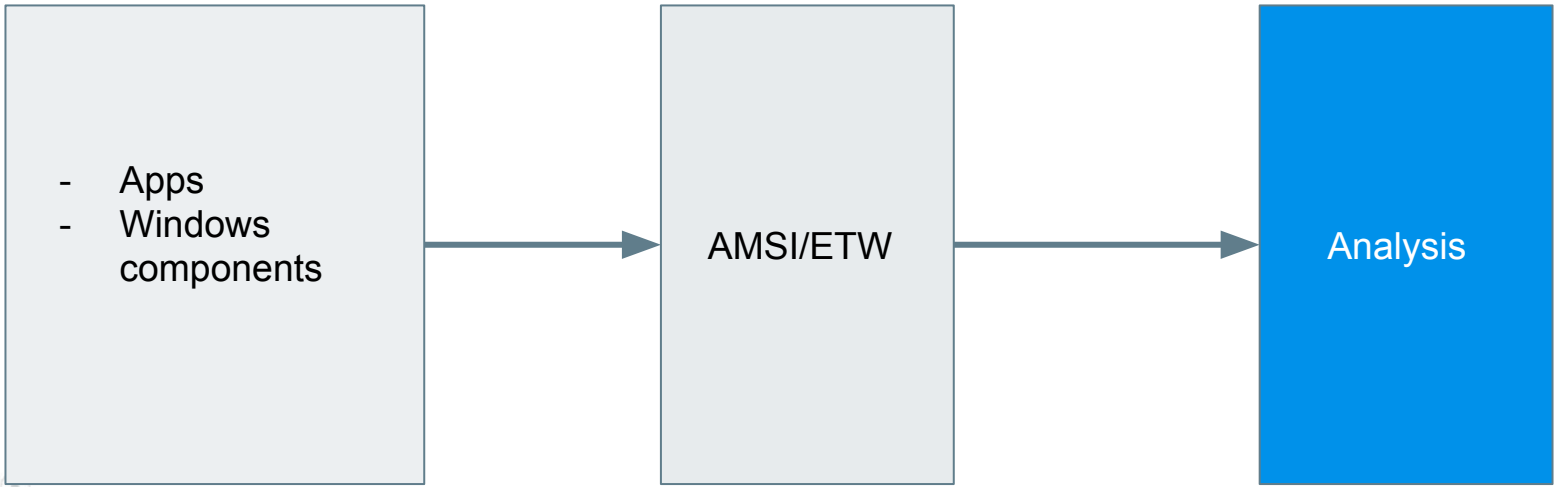
Common data collecting mechanisms

#1 - Built in Windows telemetry interfaces

- **Anti Malware Scan Interface (AMSI)**
- **Event Tracing for Windows (ETW)**
- Powershell Script Block Logging (Not covered today)

Common data collecting mechanisms

#1 - Built in Windows telemetry interfaces



Common data collecting mechanisms

#2 - Memory scanners

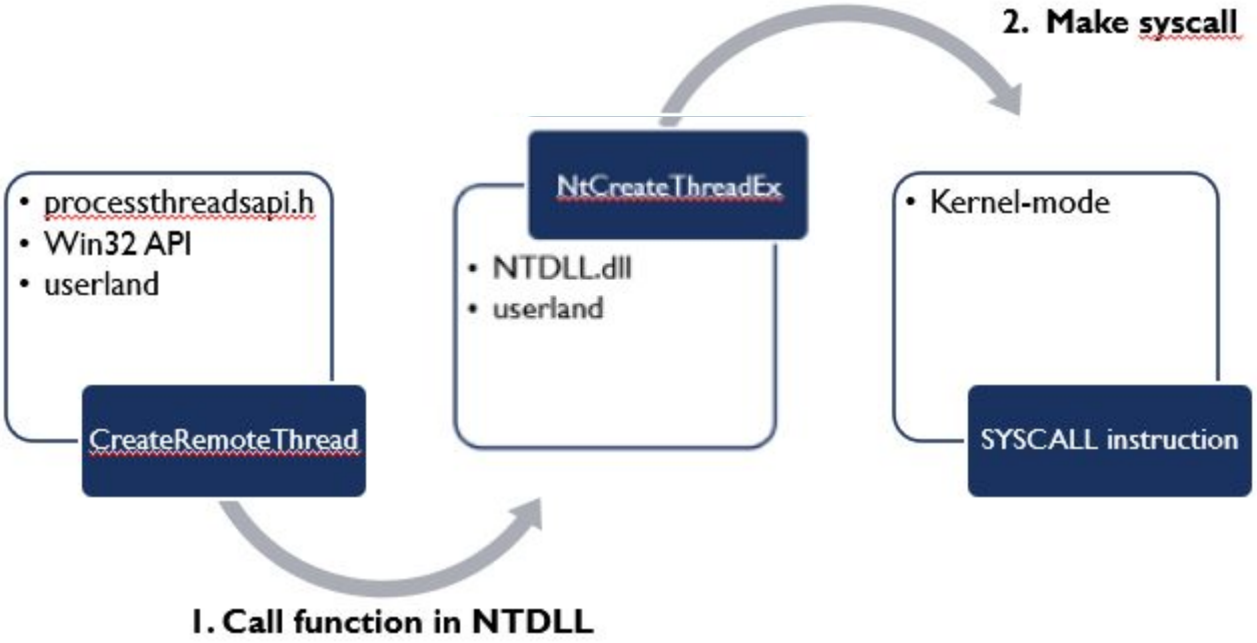
- Not changed much from traditional scanning
- Signature based (mostly)
- New techniques exist but not used during runtime as of yet
- Not constantly scanning due for performance reasons

Common data collecting mechanisms

#3 - Function hooks

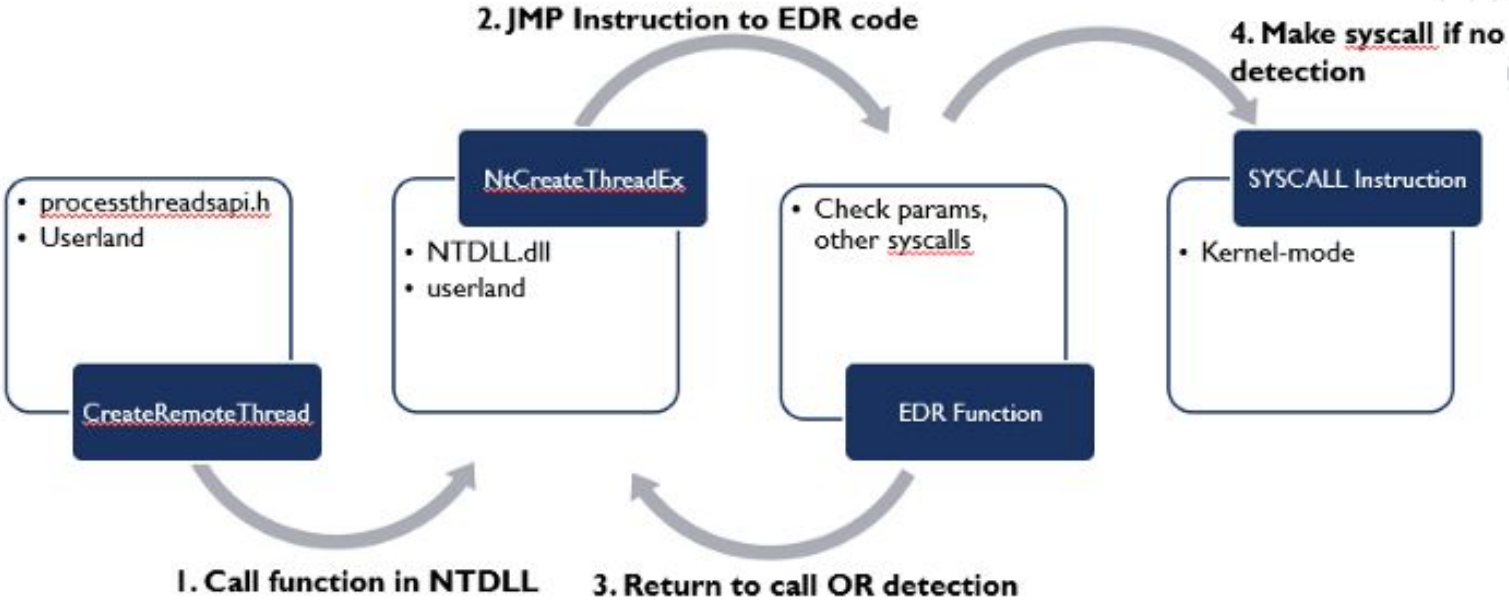
- Can tell when a function is called
- Can see parameters passed into the function

Function before being hooked



A Beginner's Guide to EDR Evasion: https://kylemistele.medium.com/a-beginners-guide-to-edr-evasion-b98cc076eb9a

Hooked function



A Beginner's Guide to EDR Evasion: https://kylemistele.medium.com/a-beginners-guide-to-edr-evasion-b98cc076eb9a

Evading modern detections

General categories of evasions

- Evade the data collector
- Disable the data collector
- Blend in

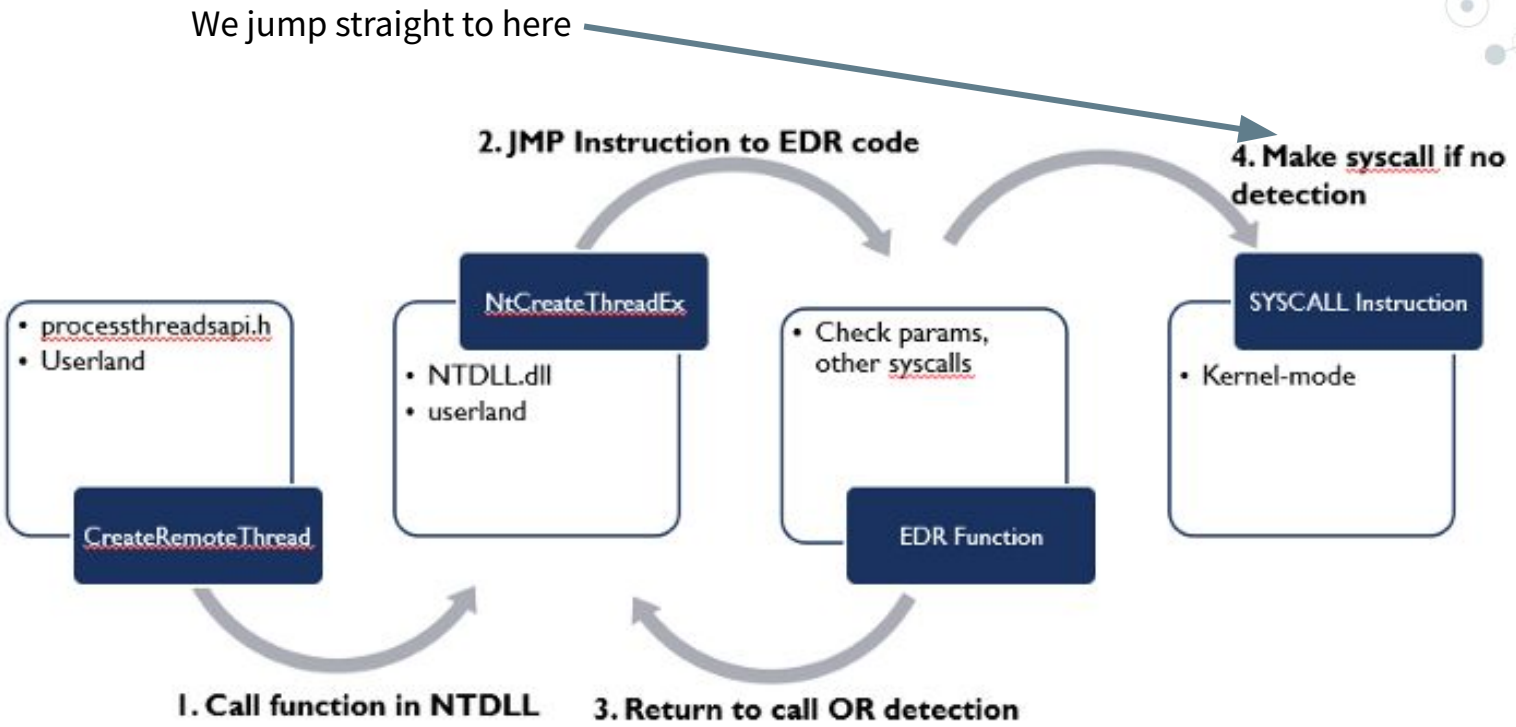
Evading the data collector

- Blue’s data collection mechanisms are intact
- Blue still can't see you (yay!)

Techniques covered

- Direct syscalls

Direct Syscalls



https://kylemistele.medium.com/a-beginners-guide-to-edr-evasion-b98cc076eb9a

Direct Syscalls

Steps

- Obtain syscall number of function to run
- Move syscall number to EAX register
- Call the syscall instruction

https://www.ired.team/offensive-security/defense-evasion/using-syscalls-directly-from-visual-studio-to-bypass-avs-edrs

Direct Syscalls

Obtaining the syscall number

- Hardcode - need to know Windows version
- Dynamically resolve from NTDLL

NtCreateThread	0x004b	0x004b
NtCreateThreadEx		
NtCreateTimer	0x0086	0x0086
NtCreateTimer2		
NtCreateToken	0x0087	0x0087

Syscall table: https://j00ru.vexillium.org/syscalls/nt/64/

Evading the data collector

- Direct Syscalls have been effective for quite a while now
- Techniques to detect them are being developed

https://passthehashbrowns.github.io/detecting-direct-syscalls-with-frida

Disabling the data collector

- Tamper with blue’s data collector
- Blue can’t see you

Techniques covered

- Patching out AMSI and ETW
- Removing function hooks from userland
- Removing kernel callbacks from kernel mode

AMSI/ETW patch

Steps

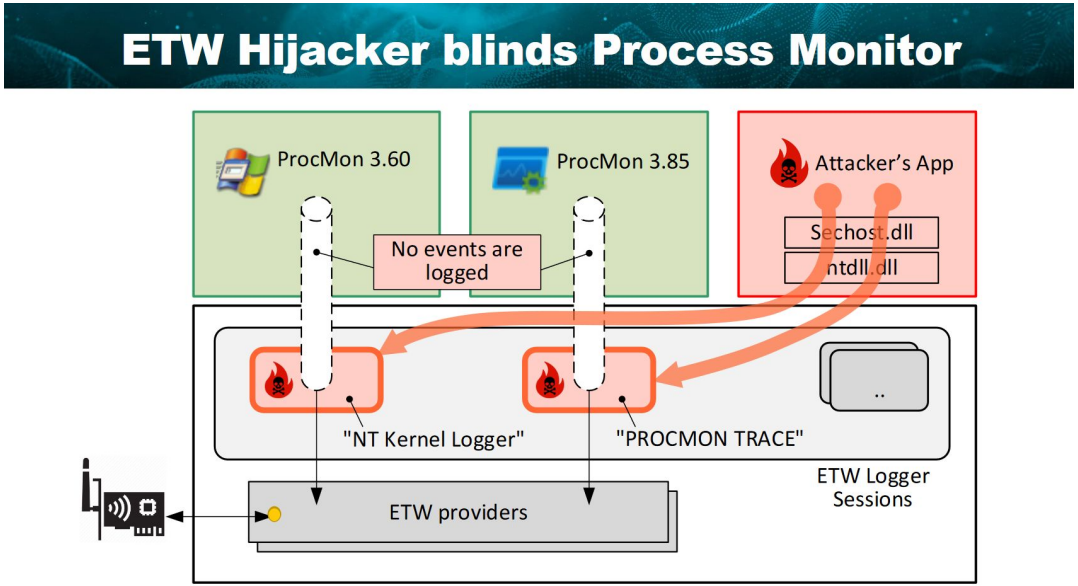
- Get address of EtwEventWrite and AmsiScanBuffer
  - In ntdll.dll and amsi.dll respectively
- Overwrite the bytes to prevent function call

```
using System;
using System.Runtime.InteropServices;

0 references
public class Amsi
{
    static byte[] x64_etw_patch = new byte[] { 0x48, 0x33, 0xC0, 0xC3 };
    static byte[] x86_etw_patch = new byte[] { 0x33, 0xc0, 0xc2, 0x14, 0x00 };
    public static Int64 x64_etw_offset = 0x1ed60;
    public static Int64 x86_etw_offset = 0x590;
    public static Int64 x64_ASB_offset = 0xcb0;
    public static Int64 x86_ASB_offset = 0x970;
    static byte[] x64 = new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3 };
    static byte[] x86 = new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC2, 0x18, 0x00 };

    1 reference
    private static string decode(string b64encoded)
    {
        return System.Text.ASCIIEncoding.ASCII.GetString(System.Convert.FromBase64String(b64encoded));
    }
```

https://blog.securityevaluators.com/creating-av-resistant-malware-part-2-1ba1784064bc

Breaking Procmon via ETW

This technique was presented at Blackhat Europe 2021 - Veni No Vidi No Vici Attacks On ETW Blind EDRs



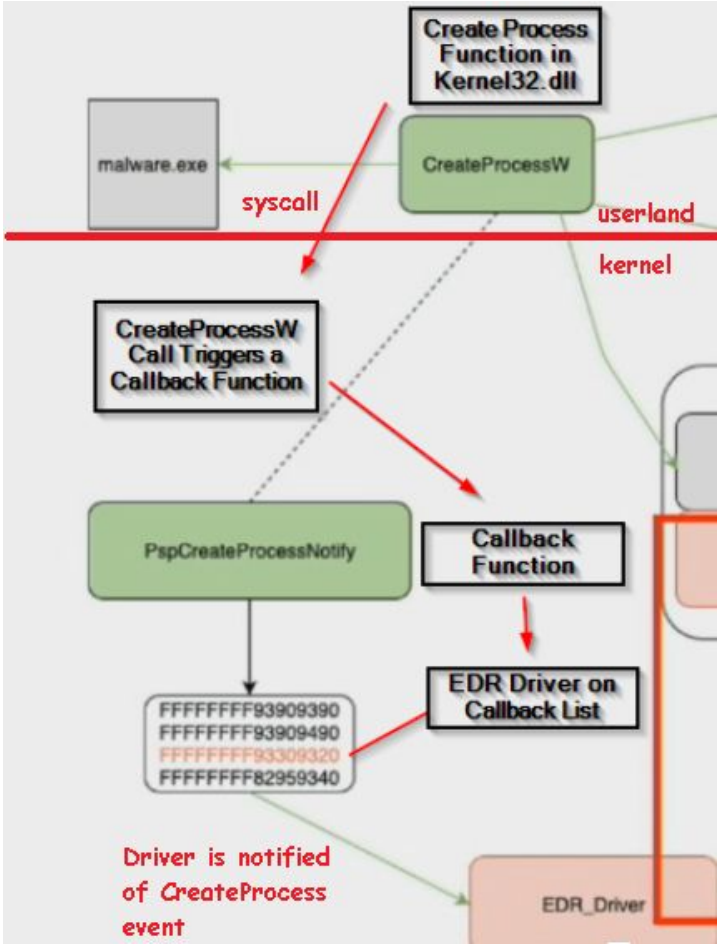
https://i.blackhat.com/EU-21/Wednesday/EU-21-Teodorescu-Veni-No-Vidi-No-Vici-Attacks-On-ETW-Blind-EDRs.pdf

Understanding how EDRs hook

Understanding kernel mode vs user mode

- Everything you do is in user mode
- When processes need the kernel to do things it makes a syscall
- You **shouldn't** be able to modify anything in kernel from user mode (PatchGuard)
- **You can register kernel "callbacks" using a driver/minifilter to notify on certain kernel events**

https://synzack.github.io/Blinding-EDR-On-Windows/



https://synzack.github.io/Blinding-EDR-On-Windows/

Understanding how EDRs hook

Callbacks vs Hooks

	Kernel Callback	Function hook
Can be applied to any function	No - existing list of supported callbacks	Yes
Can perform analysis and detection	No - notifies driver of call	Yes - intercepts function parameters and can execute its own code
Can be removed from userland	No - only from kernel	Yes

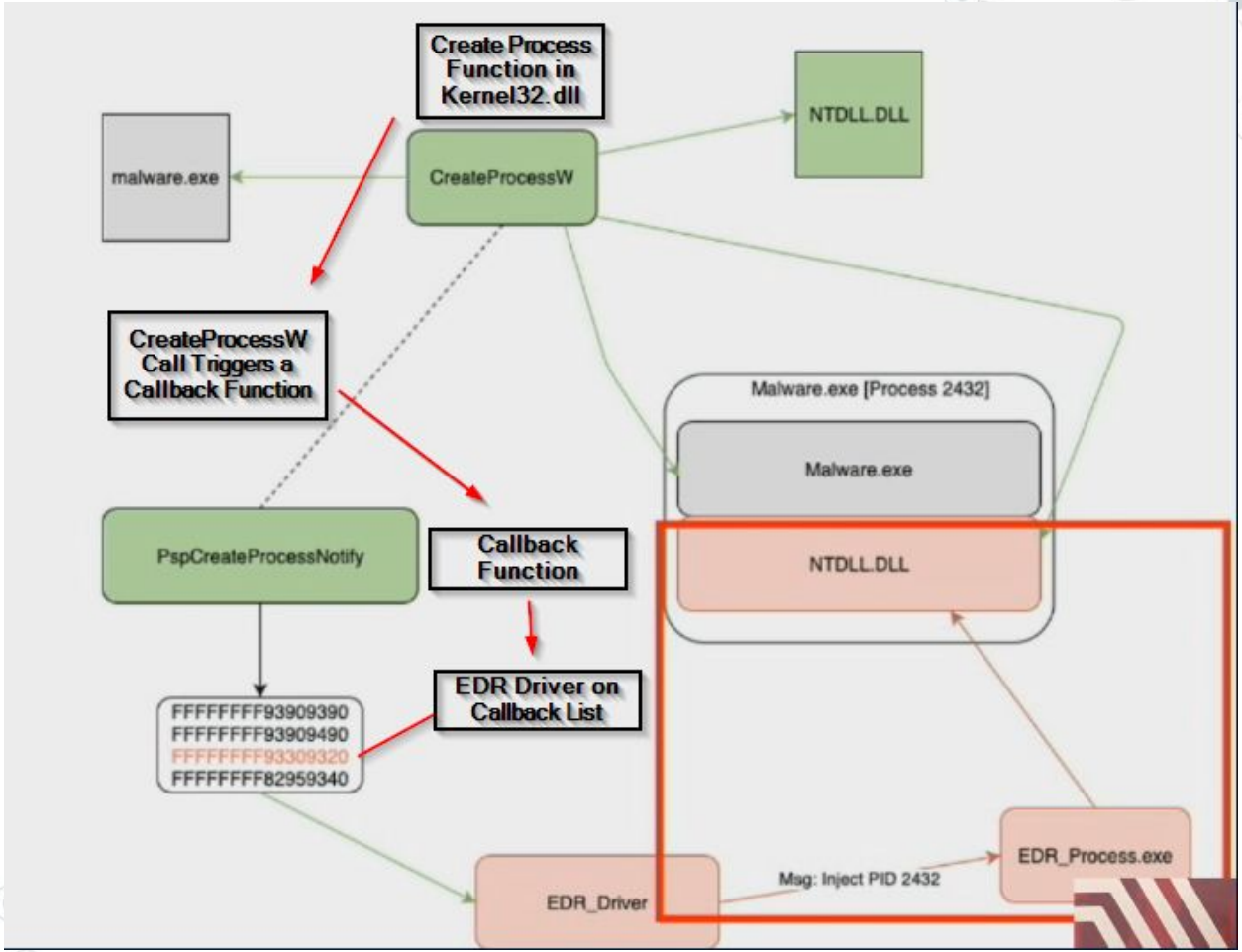
https://synzack.github.io/Blinding-EDR-On-Windows/

Understanding how EDRs hook

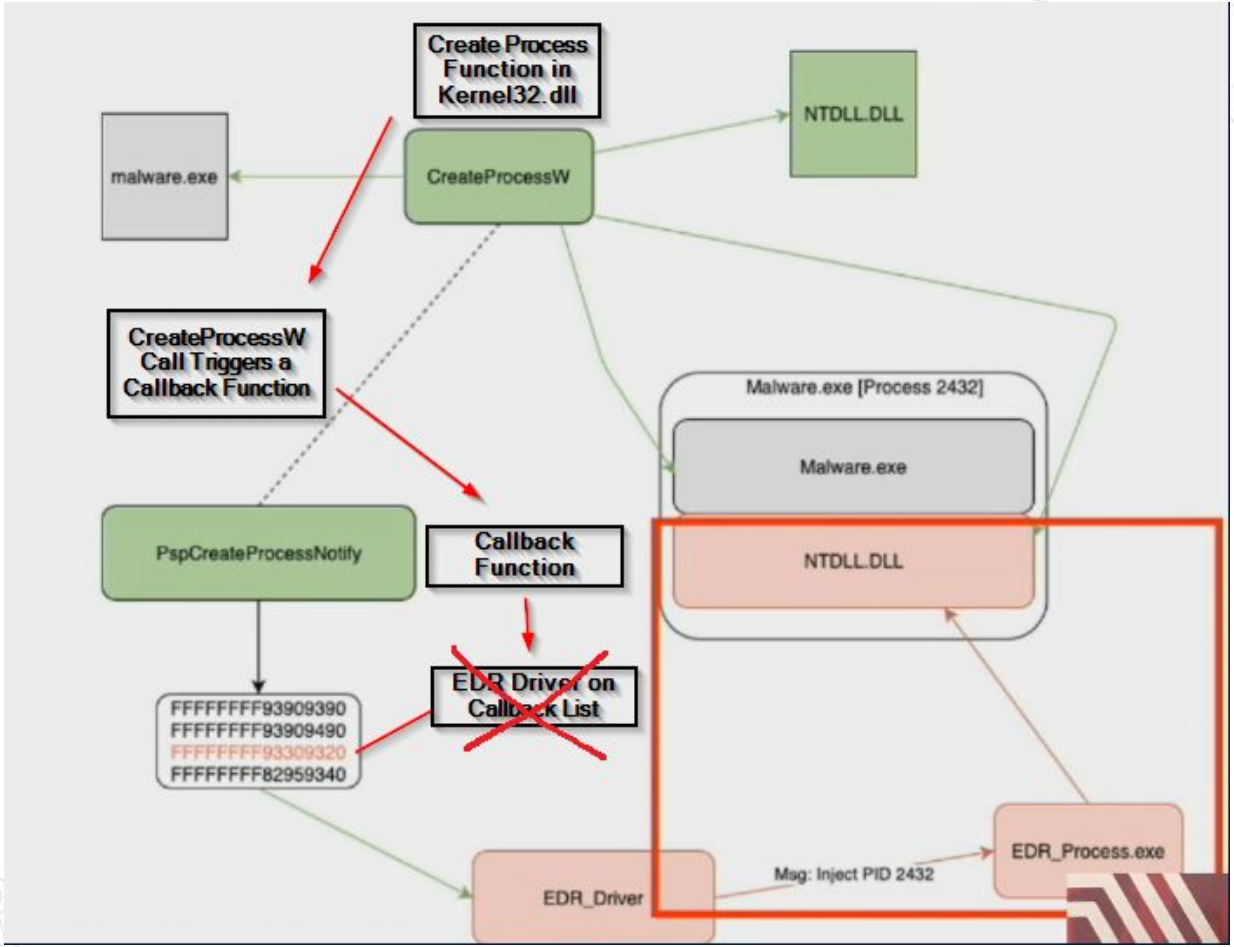
EDR process flow

- EDR driver registers a kernel callback to notify on process creation
- EDR driver receives notification of process creation
- On notification, the EDR driver instructs the EDR process to inject its DLL into the new process
- The DLL applies the userland hook to the process it was injected into

https://synzack.github.io/Blinding-EDR-On-Windows/



https://synzack.github.io/Blinding-EDR-On-Windows/



https://synzack.github.io/Blinding-EDR-On-Windows/

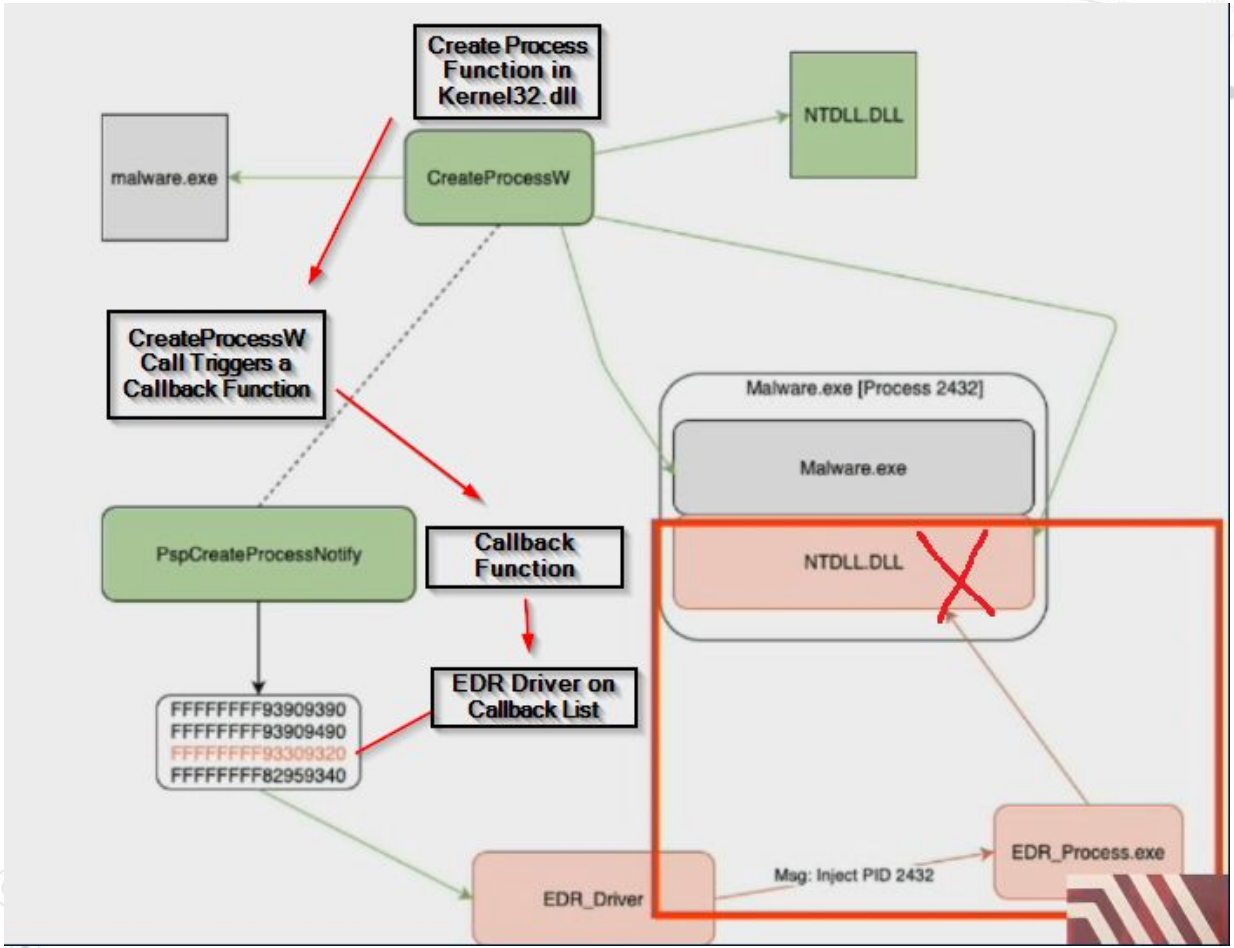
Removing the kernel callback

Remember, user mode applications aren't supposed to be able to directly touch the kernel

- Need elevated privileges to install a driver
- Driver locates callback array in memory
- Driver patches callback array to remove EDR callback
- If EDR doesn’t receive the callback it doesn’t inject the hook

What if we don’t have local admin?

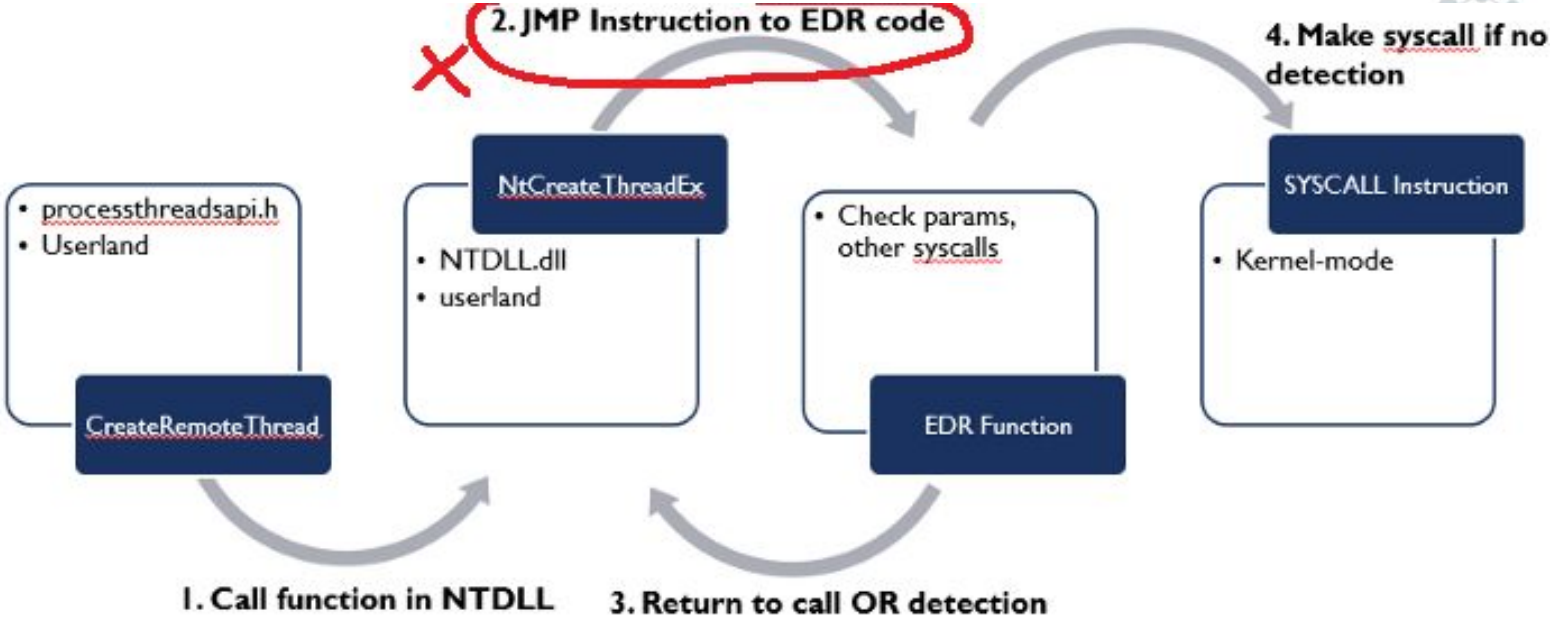
https://blog.nviso.eu/2021/10/21/kernel-karnage-part-1/



https://synzack.github.io/Blinding-EDR-On-Windows/

Removing function hooks from userland

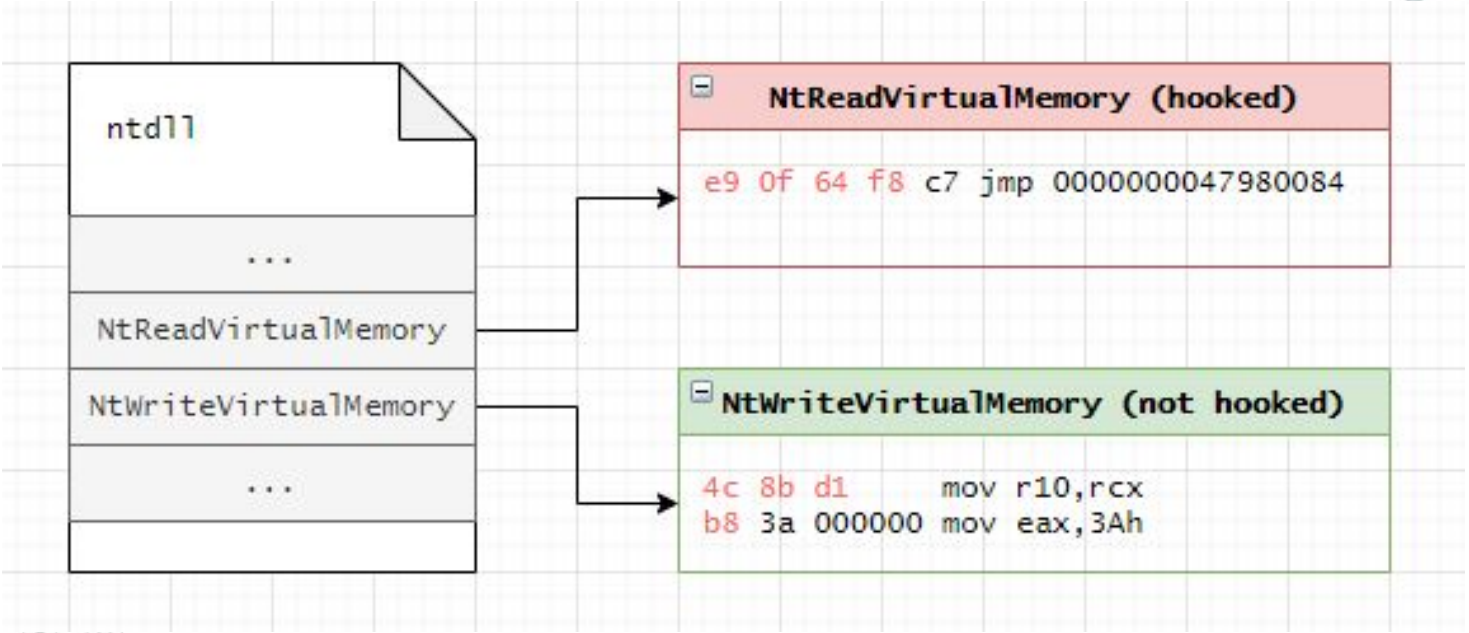
Locating the hook



https://kylemistele.medium.com/a-beginners-guide-to-edr-evasion-b98cc076eb9a

Removing function hooks from userland

Hooked vs unhooked bytes



https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++

https://www.ired.team/offensive-security/defense-evasion/detecting-hooked-syscall-functions

Removing function hooks from userland

Methods

- Overwrite the entire .text section of NTDLL with a clean copy from disk
- Locate the hook and overwrite it with the original bytes

Goal is to remove the bytes that pass execution to the EDR

https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++

EDRs without their hooks

Demo: Unhooking

Disabling the data collector

- Tampering with defenses itself can become an IOC
- Some (or most) of these evasions if spotted by a defender can be flagged

Blending in

- Blue’s data collection mechanisms are intact
- Blue can see you
- Blue doesn’t think you are bad

Techniques covered

- Behaviour
- Sleep protections - hide memory indicators

Blending in

Memory indicators often flagged by recent detection projects and EDR

- Known malware signatures (BeaconEye)
- Abnormal PE structures (Moneta, PEsieve)
- Anomalies in call stack (Syscalls not coming from NTDLL)
- Modified AMSI/ETW or function hooks

Call stack anomaly detection rule: https://github.com/elastic/detection-rules/issues/1535

Blending in

Recent techniques to blend in in memory

- Clearing PE headers left by RunPE techniques e.g. donut, RDI
  - Can be taken further, Position Independent Code (PIC) the payload itself

BruCON PIC malware - https://www.youtube.com/watch?v=8UCBvvJZw2U
Stack Spoofing - https://github.com/mgeeky/ThreadStackSpoofer
Heap encryption - https://www.arashparsa.com/hook-heaps-and-live-free/

Blending in

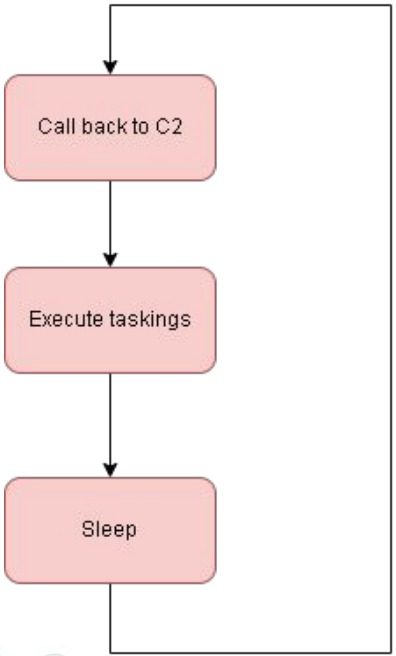
Recent techniques to blend in in memory

- Sleep protection
  - Protect asynchronous beaconing agents while they sleep
    - Stack spoofing
    - Memory region protection
    - Obfuscation and encryption e.g. shellcode, heap etc.
    - Restore defenses (AMSI/ETW/Hooks)

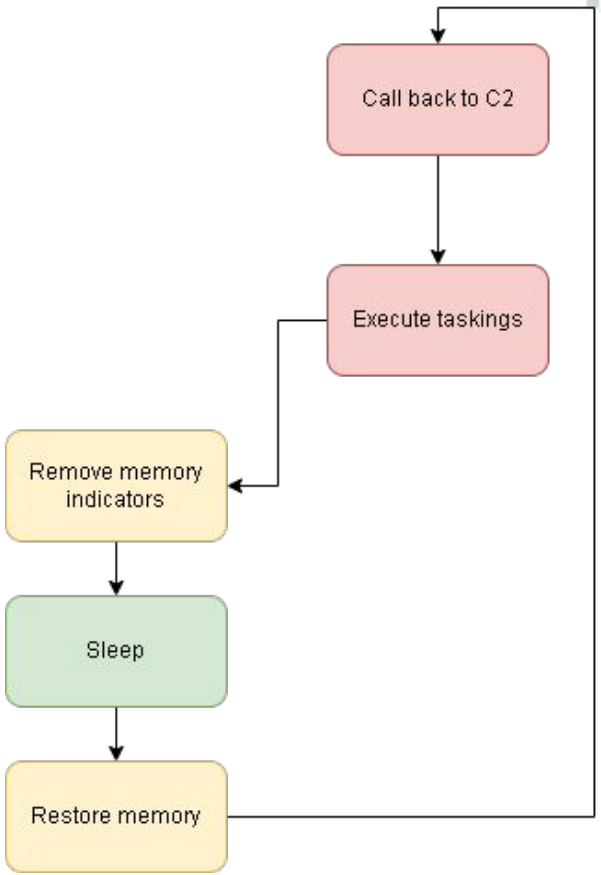
BruCON PIC malware - https://www.youtube.com/watch?v=8UCBvvJZw2U
Stack Spoofing - https://github.com/mgeeky/ThreadStackSpoofer
Heap encryption - https://www.arashparsa.com/hook-heaps-and-live-free/

Blending in

Beacon execution flow



Sleep protection



F-Secure GARGOYLE - https://labs.f-secure.com/blog/experimenting-bypassing-memory-scanners-with-cobalt-strike-and-gargoyle/

Demo: Sleep protection

Blending in

What is the “normal” in the target environment?

- Who uses what processes for what, when etc.
- Situational Awareness

Normal windows behavior

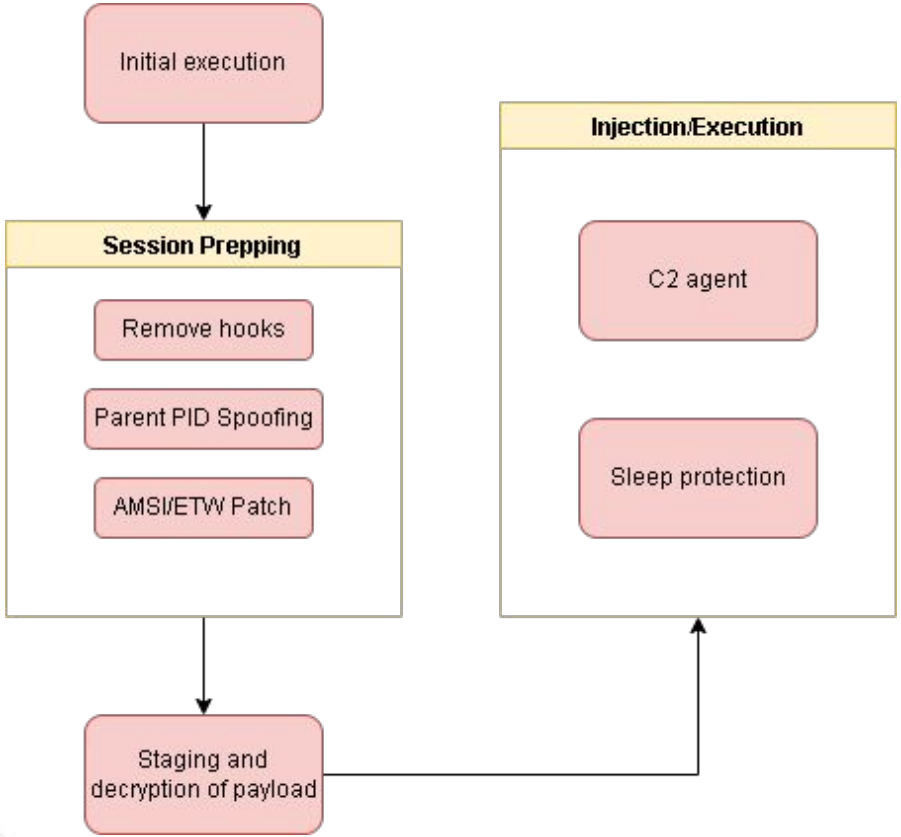
- Parent child relationships
- Process privileges
- Built in windows processes have well known behaviour

Behaviour ⇒ Techniques

Some techniques to emulate “known good” behaviour

- PPID spoofing
- Cert stealing (can be double edged sword)
- Custom C2 channels based on target behaviour
- DLL proxying (loading technique that is good for evasion)
- .NET and COM are great
- Really anything you can do that makes your “disguise” believable

Blending in - Payload execution



tl;dr

Evasion	Evades hooks	Evades telemetry	Evades anomaly detection	Evades memory scanning
AMSI/ETW patch	No	Yes	No	No
Removing hooks	Yes	No	No	No
Direct Syscalls	Yes	No	No	No
Encryption while sleeping	No	No	Yes	Yes
Stack Spoofing	No	No	Yes	Yes (depends)
D/Invoke	Yes	No	No	No
PPID Spoofing	No	No	Yes	No
PIC	No	No	Yes	Yes (depends)

Things to note in current implementations

- Specialized detection techniques are being developed
- Some techniques leave their own IOCs
- Combination of techniques can reduce IOCs
- Operator actions matter a lot

Detecting syscalls - https://passthehashbrowns.github.io/detecting-direct-syscalls-with-frida
Sleep protections - https://github.com/mgeeky/ShellcodeFluctuation#how-do-i-use-it

Conclusion

- Cat and mouse game
- Blue has shifted their detection approach
  - Known bad ⇒ Not known normal
- Red has shifted their tradecraft accordingly
- Techniques can be combined based on target
- Both red and blue tradecraft are advancing

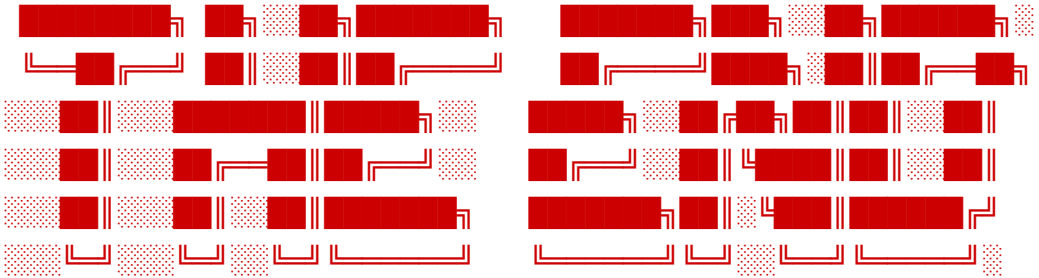
Useful resources

In depth explanations

- https://www.ired.team/offensive-security/defense-evasion ⇐ general red team wiki
- https://thewover.github.io/Dynamic-Invoke/ ⇐ d/invoke explained
- https://blog.nviso.eu/2021/10/21/kernel-karnage-part-1/ ⇐ removing kernel callbacks
- https://adamsvoboda.net/sleeping-with-a-mask-on-cobaltstrike/ ⇐ sleep protection
- https://www.arashparsa.com/hook-heaps-and-live-free/ ⇐ heap encryption
- https://vxug.fakedoma.in/papers/VXUG/Exclusive/HellsGate.pdf ⇐ Dynamically resolving syscall numbers

Demo files:

- https://github.com/CodeXTF2/evasion-adventures-files ⇐ Slides and unhooking demo
- https://github.com/mgeeky/ShellcodeFluctuation ⇐ Sleep protection demo



Special thanks to:
Sunny
Guo Gen

QnA