DefCamp where hacking & security collide Cluj Napoca

# Many thanks to our sponsors and partners!

Powered by orange

## PLATINUM SPONSORS

BIT SENTINEL

D3 CYBER

CYBER LIFE HACKS

## HACKING VILLAGE PARTNERS

cyber EDU

electron

HACKOUT Portalul Atacurilor Cibernetice

## SILVER SPONSORS

efect

PFG FINANCE

wantsome the friendly IT academy

## MOBILITY PARTNER

TOYOTA Cluj-Napoca prin Profi Auto

## COMMUNITY & MEDIA PARTNERS

DIRECTORATUL NAȚIONAL DE SECURITATE CIBERNETICĂ

CLUJ IT

UNIVERSITATEA BABEȘ-BOLYAI
BABEȘ-BOLYAI TUDOMÁNYEGYETEM
BABEȘ-BOLYAI UNIVERSITÄT
BABEȘ-BOLYAI UNIVERSITY
TRADITIO ET EXCELLENTIA

BRCC British Romanian Chamber of Commerce

ISACA Trust in, and value from, information systems Romania Chapter

(ISC)² CHAPTER ROMANIA

Cloud Security Alliance Official Chapter Romania Chapter CSA

CARTEA DALIEI

x86 GENERATION

WOMEN 4CYBER EUROPEAN CYBER SECURITY ORGANISATION ROMANIA

ȘCOALA INFORMALĂ DE IT®

TSM TODAY SOFTWARE MAGAZINE

DevExperience

ITCAMP

MOBZINE.RO

SecurityPatch

ROTSA The highway of Romanian Tech Startups

techcelerator

Global Leader
In Cybersecurity

# Bitdefender®

# Code Injection via Arbitrary Pointer Overwrite

Bitdefender

# Introduction – Eduard Muresan

- Security Researcher @ Bitdefender
- Passionate about low level systems programming
- Interested in injection techniques
- Mythic raider at night

Bitdefender

# Agenda

- ↳ Process Injection Background
- ↳ C.I.A.P.O. Methodology
- ↳ Executable Pointer Examples
- ↳ Mitigations
- ↳ Demo
- ↳ Q&A

Bitdefender

# What is process injection

Bitdefender

# Process injection

A method used by malware to execute arbitrary code within the address space of a separate live process.

Bitdefender

# Why process injection

Bitdefender

# Process injection

Execution via process injection might evade process-based detections from security products.

Bitdefender

# Injection building blocks

Bitdefender

# Injection building blocks

Bitdefender

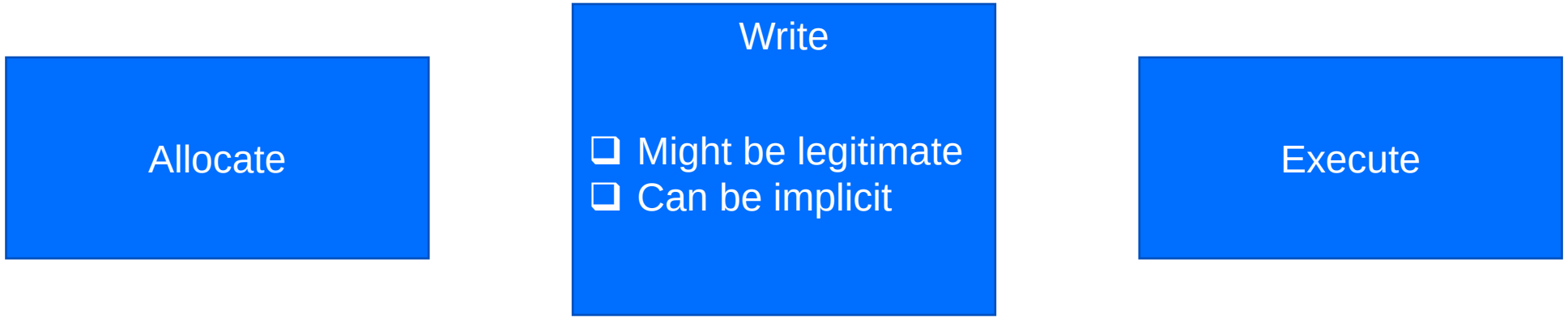
# Injection building blocks

Bitdefender

# Injection building blocks

Bitdefender

# Injection building blocks



Bitdefender

# Injection building blocks

Bitdefender

# Piecing everything together

Bitdefender

# Piecing everything together

```
void RemoteThreadInjection(HANDLE Process) {
    // Pretend this is a fancy shellcode.
    static const BYTE shellcode[] = { 0xC3 };

    // Reserve space for the payload.
    void* payload = VirtualAllocEx(Process, NULL, sizeof(shellcode), MEM_RESERVE |
       MEM_COMMIT,    PAGE_EXECUTE_READWRITE);

    // Write the payload to the allocated space.
    WriteProcessMemory(Process, payload, shellcode, sizeof(shellcode), NULL);

    // Trigger the payload.
    CreateRemoteThread(Process, NULL, 0, payload, NULL, 0, NULL);
}
```

Allocate

Write

Execute

Bitdefender

# Piecing everything together

```
void RemoteThreadInjection(HANDLE Process) {
    // Pretend this is a fancy shellcode.
    static const BYTE shellcode[] = { 0xC3 };

    // Reserve space for the payload.
    void* payload = VirtualAllocEx(Process, NULL, sizeof(shellcode), MEM_RESERVE |
        MEM_COMMIT,    PAGE_EXECUTE_READWRITE);

    // Write the payload to the allocated space.
    WriteProcessMemory(Process, payload, shellcode, sizeof(shellcode), NULL);

    // Trigger the payload.
    CreateRemoteThread(Process, NULL, 0, payload, NULL, 0, NULL);
}
```

Bitdefender

# Change the execute primitive

```
void APCInjection(HANDLE Process, HANDLE Thread) {
    // Pretend this is a fancy shellcode.
    static const BYTE shellcode[] = { 0xC3 };

    // Reserve space for the payload.
    void* payload = VirtualAllocEx(Process, NULL, sizeof(shellcode), MEM_RESERVE | MEM_COMMIT,
        PAGE_EXECUTE_READWRITE);

    // Write the payload to the allocated space.
    WriteProcessMemory(Process, payload, shellcode, sizeof(shellcode), NULL);

    // Trigger the payload.
    QueueUserAPC(payload, Thread, NULL);
}
```

Bitdefender

# The problem with execute primitives

Bitdefender

# The problem with execute primitives

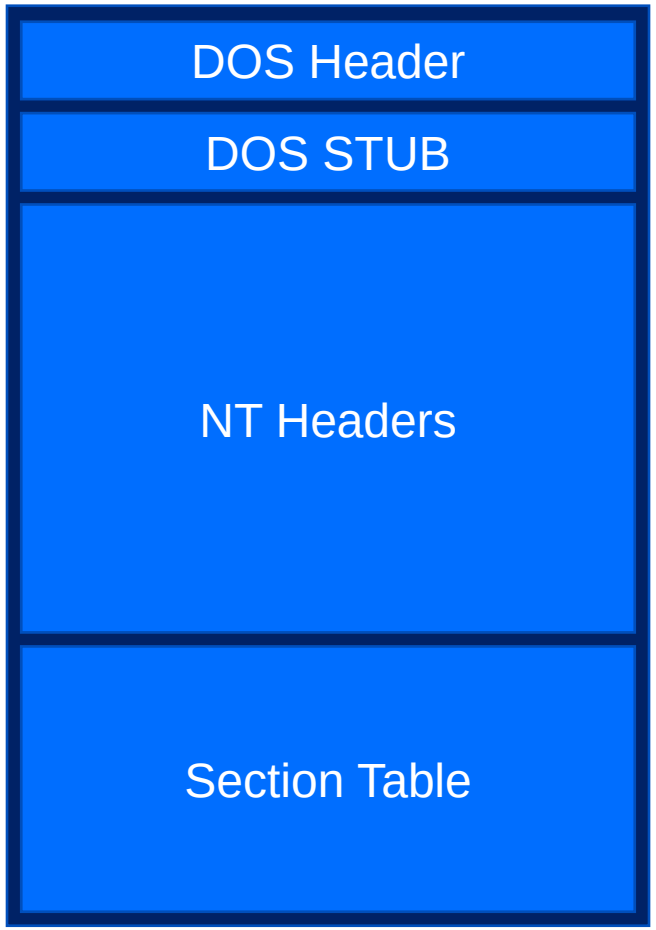
- ❑ Highly monitored
- ❑ Can be blocked
- ❑ Might be traced back to the attacker
- ❑ Becomes a cat and mouse game

Bitdefender

# Remove the execute primitive?

Bitdefender

# The PE file format



Bitdefender

# The PE file format

Bitdefender

# The PE file format

Bitdefender

# Thread Local Storage Injection

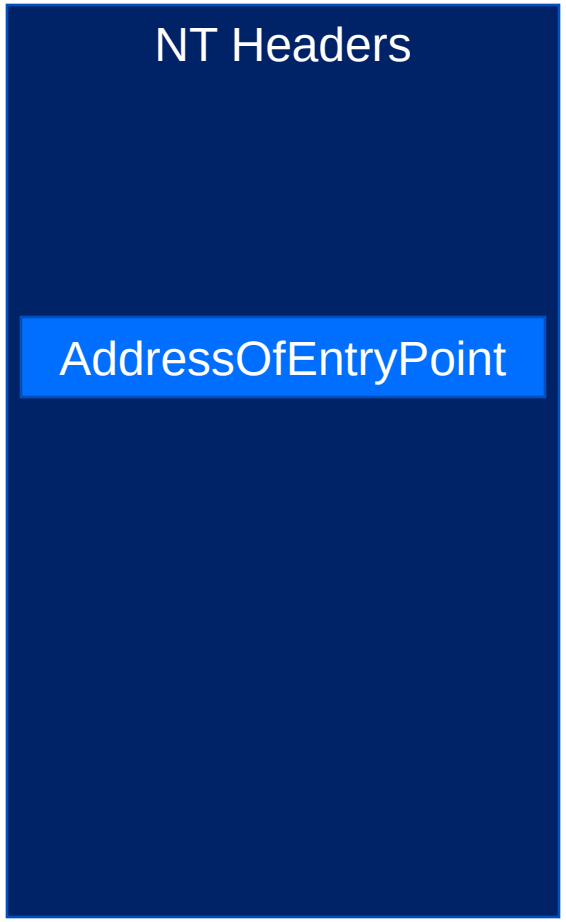

```
typedef struct _IMAGE_TLS_DIRECTORY64 {
    ULONGLONG StartAddressOfRawData;
    ULONGLONG EndAddressOfRawData;
    ULONGLONG AddressOfIndex;         // PDWORD
    ULONGLONG AddressOfCallBacks;     // PIMAGE_TLS_CALLBACK *;
    DWORD SizeOfZeroFill;
    union {
        DWORD Characteristics;
        struct {
            DWORD Reserved0 : 20;
            DWORD Alignment : 4;
            DWORD Reserved1 : 8;
        } DUMMYSTRUCTNAME;
    } DUMMYUNIONNAME;
} IMAGE_TLS_DIRECTORY64;
```

Bitdefender

# The PE file format

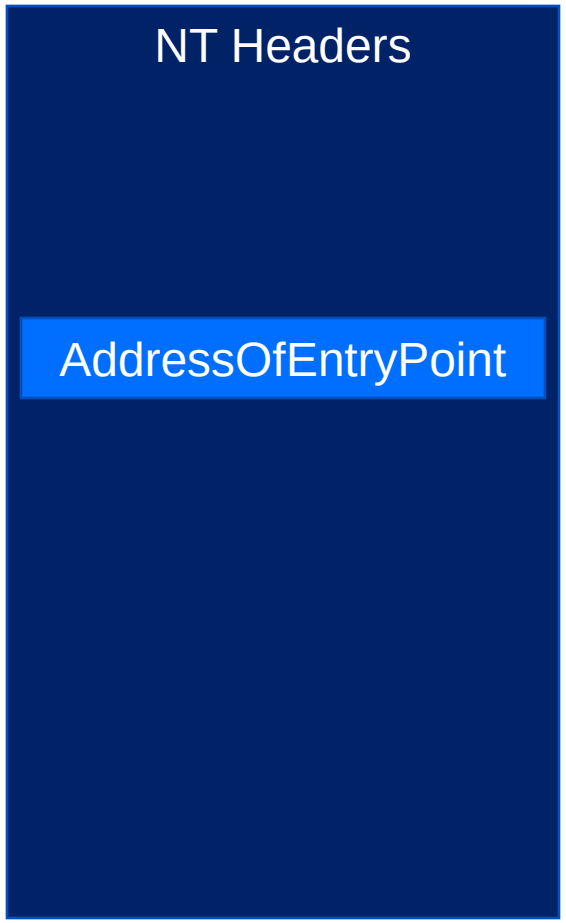
Bitdefender

# The PE file format



Bitdefender

# Entry Point Injection



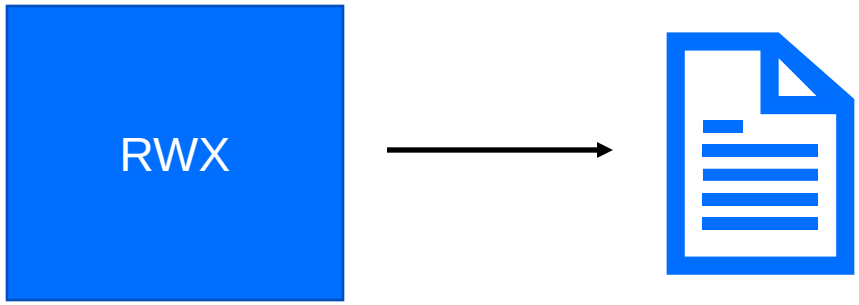

```
typedef struct _PEB_LDR_DATA {
    BYTE       Reserved1[8];
    PVOID      Reserved2[3];
    LIST_ENTRY InMemoryOrderModuleList;
} PEB_LDR_DATA, * PPEB_LDR_DATA;

typedef struct _LDR_DATA_TABLE_ENTRY {
    PVOID Reserved1[2];
    LIST_ENTRY InMemoryOrderLinks;
    PVOID Reserved2[2];
    PVOID DllBase;
    PVOID EntryPoint;
    PVOID Reserved3;
    UNICODE_STRING FullDllName;
    [snip]
} LDR_DATA_TABLE_ENTRY, * PLDR_DATA_TABLE_ENTRY;
```

Bitdefender

# Process Mockingjay

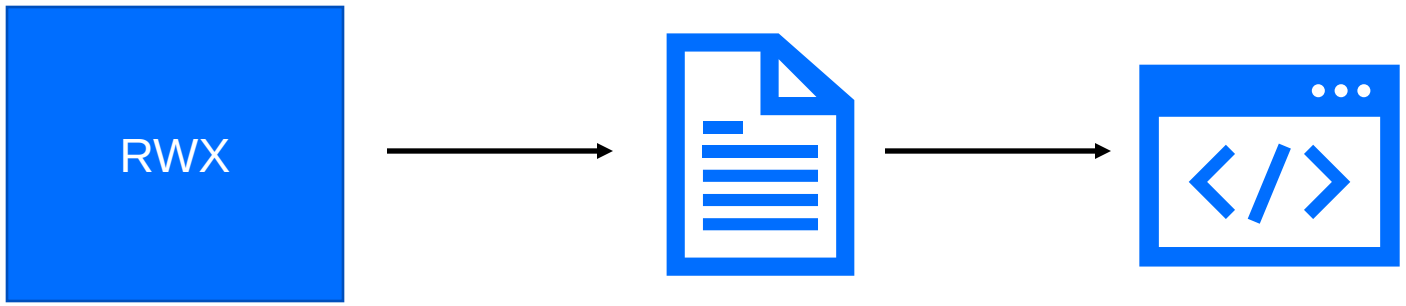
Bitdefender

# Process Mockingjay



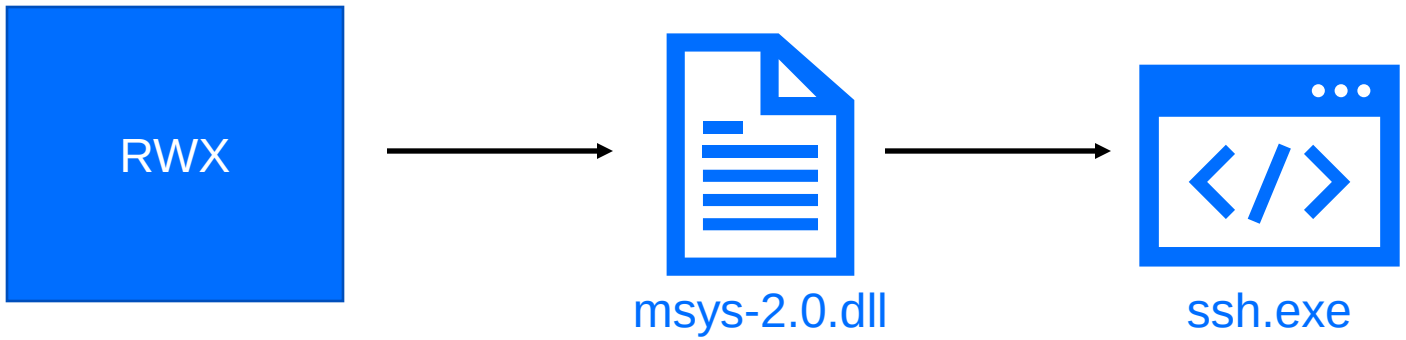
Bitdefender

# Process Mockingjay



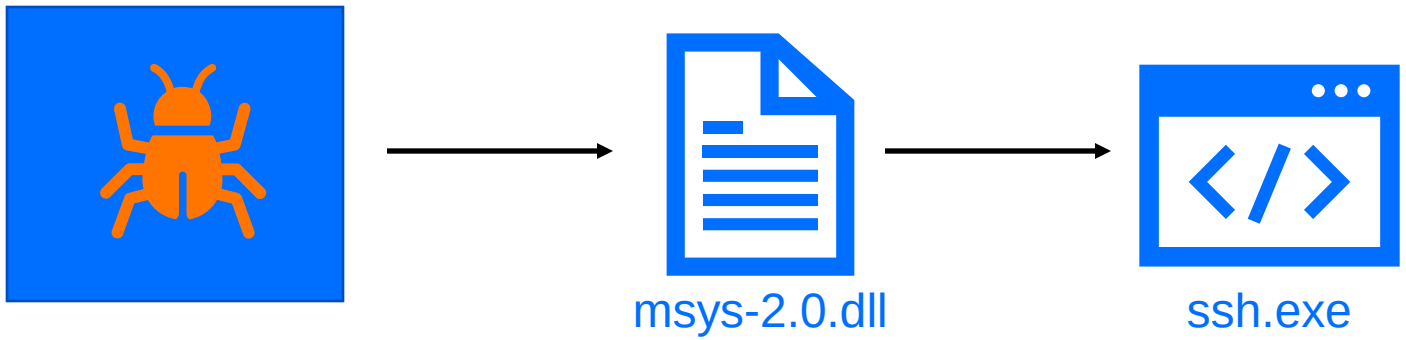
Bitdefender

# Process Mockingjay



Bitdefender

# Process Mockingjay



Bitdefender

# The problem with specific pointers

Bitdefender

# The problem with specific pointers

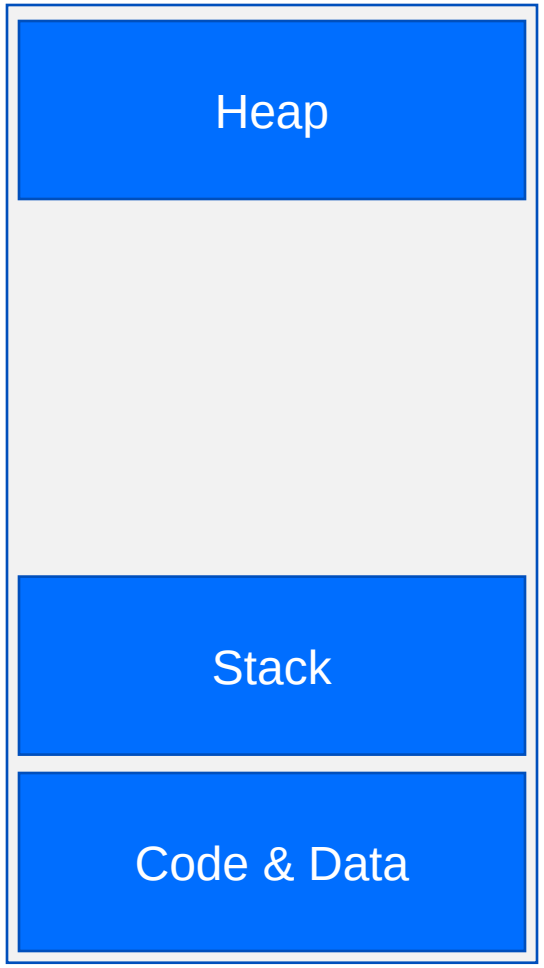
- ❑ Highly monitored
- ❑ Execution can be blocked
- ❑ Becomes a cat and mouse game

Bitdefender

# C.I.A.P.O. Methodology

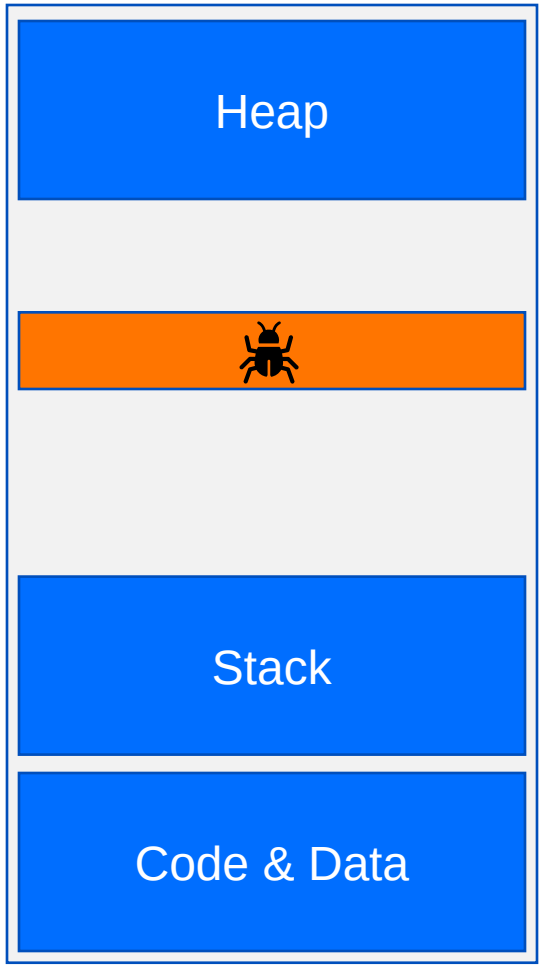
Bitdefender

# C.I.A.P.O. Methodology



Bitdefender

# C.I.A.P.O. Methodology

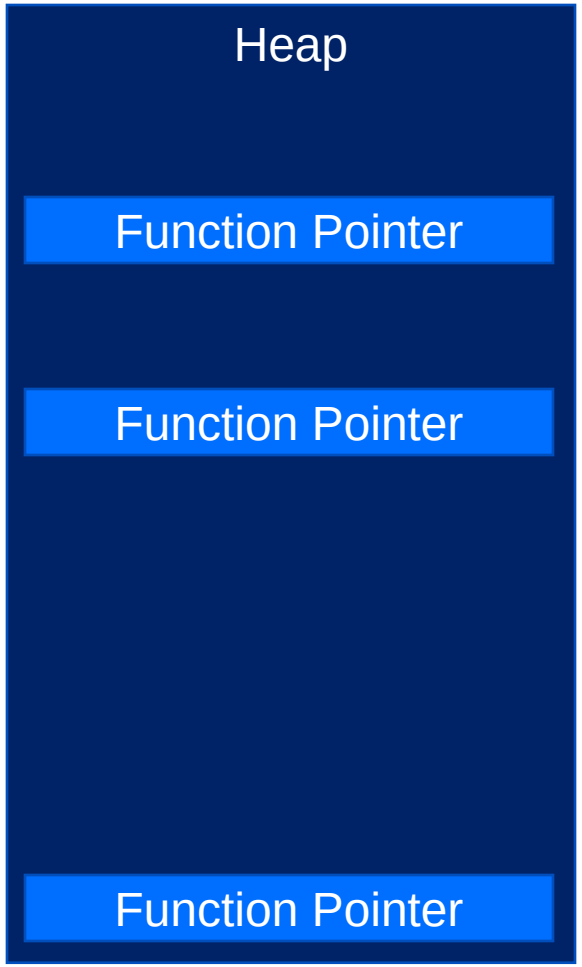


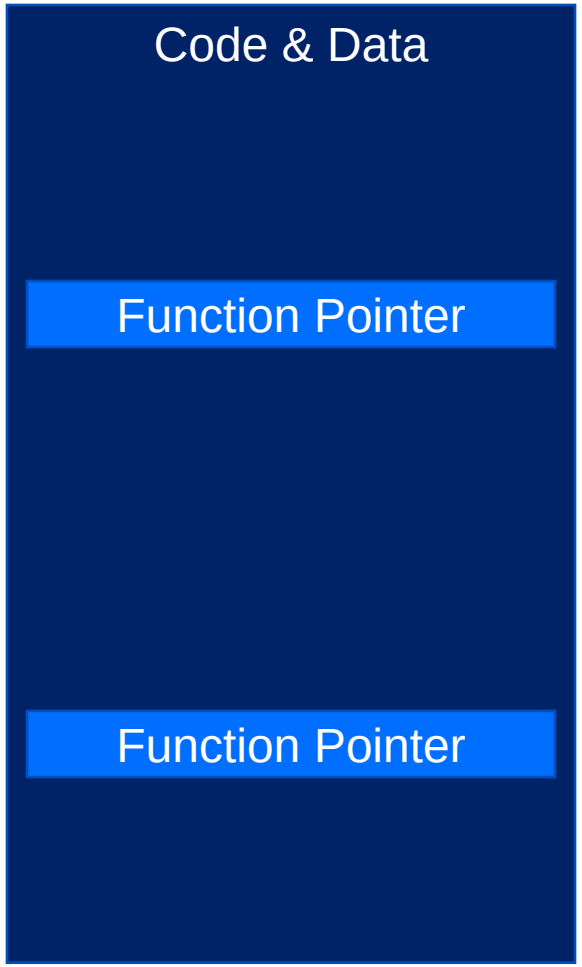
Bitdefender

# C.I.A.P.O. Methodology

Bitdefender

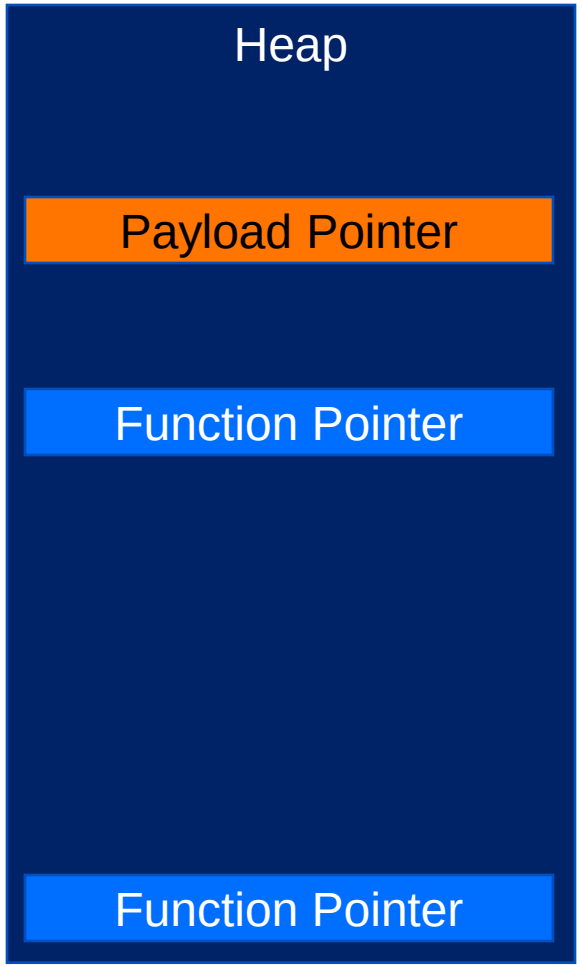
# C.I.A.P.O. Methodology

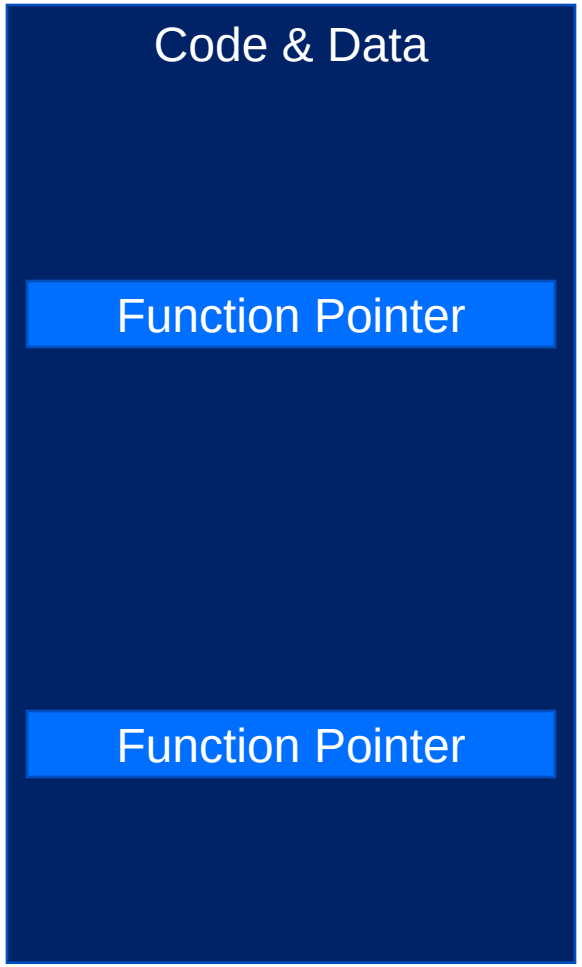


Bitdefender

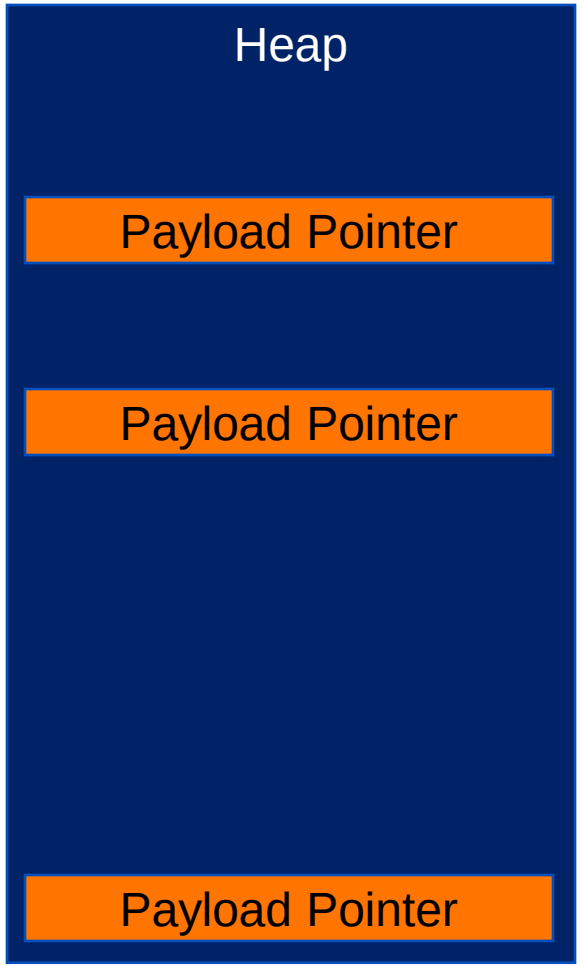
# C.I.A.P.O. Methodology

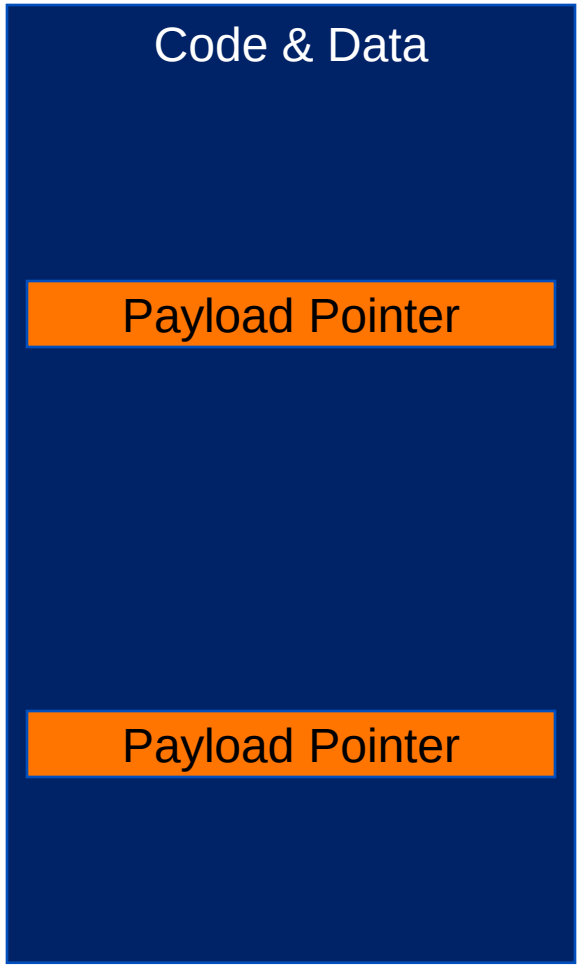


Bitdefender

# C.I.A.P.O. Methodology



Bitdefender

# All your pointer are belong to us

Bitdefender

# Example

```
static void (*pSleep)(DWORD);

int main() {
    pSleep = GetProcAddress(
        GetModuleHandleA("kernel32.dll"),
        "Sleep"
    );

    while (TRUE) {
        pSleep(1000);
    }

    return 0;
}
```

Bitdefender

# Example

```
static void (*pSleep)(DWORD);

int main() {
    pSleep = GetProcAddress(
        GetModuleHandleA("kernel32.dll"),
        "Sleep"
    );

    while (TRUE) {
        pSleep(1000);
    }

    return 0;
}
```

```
sub     rsp, 28h
lea     rcx, ModuleName ; "kernel32.dll"
call    cs:__imp_GetModuleHandleA
mov     rcx, rax        ; hModule
lea     rdx, ProcName   ; "Sleep"
call    cs:__imp_GetProcAddress
mov     cs:pSleep, rax

loc_140001028:
mov     ecx, 3E8h
call    rax
mov     rax, cs:pSleep
jmp     short loc_140001028
```

Bitdefender

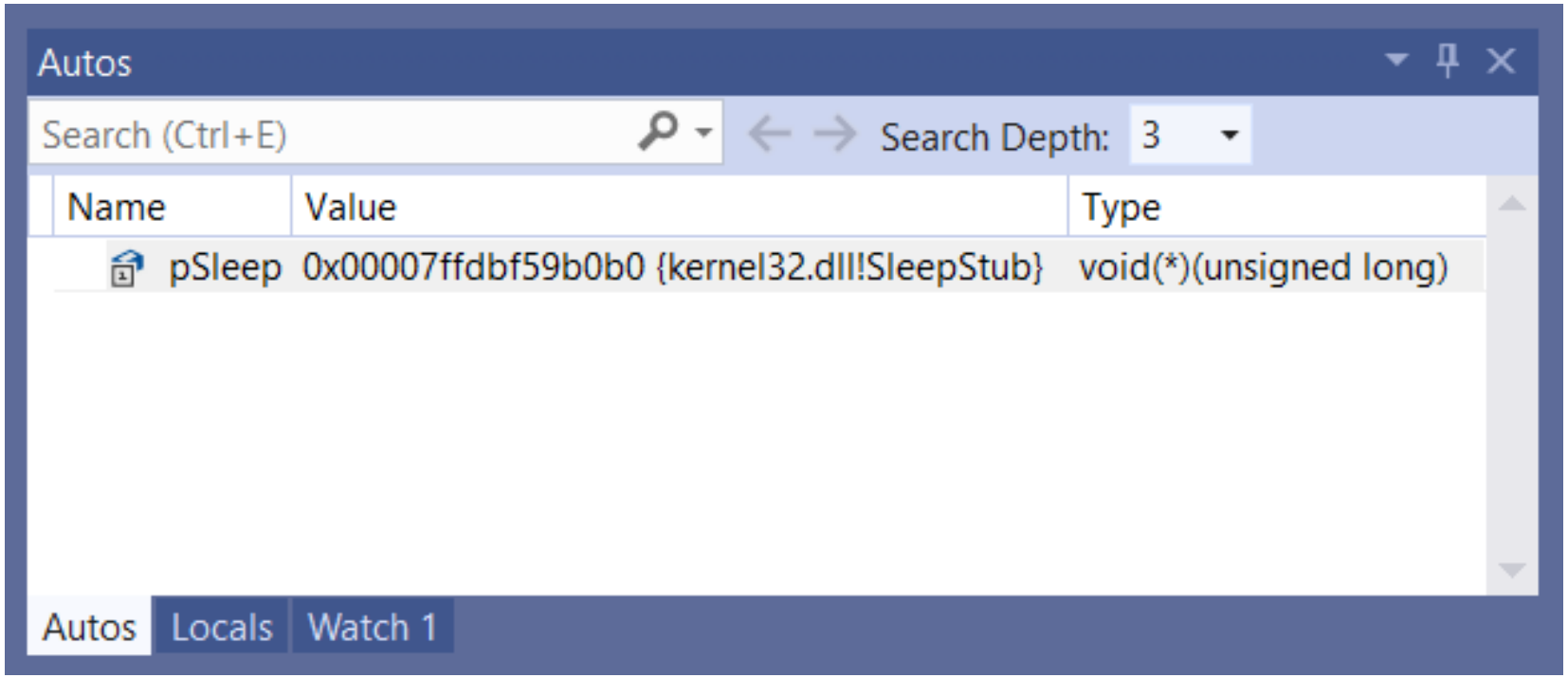
# Indirect Branches

- Dynamically determine the target address at runtime
- Generated by the compiler in most programs
- Almost guaranteed to be executed at some point

```
sub     rsp, 28h
lea     rcx, ModuleName ; "kernel32.dll"
call    cs:__imp_GetModuleHandleA
mov     rcx, rax        ; hModule
lea     rdx, ProcName   ; "Sleep"
call    cs:__imp_GetProcAddress
mov     cs:pSleep, rax

loc_140001028:
mov     ecx, 3E8h
call    rax
mov     rax, cs:pSleep
jmp     short loc_140001028
```

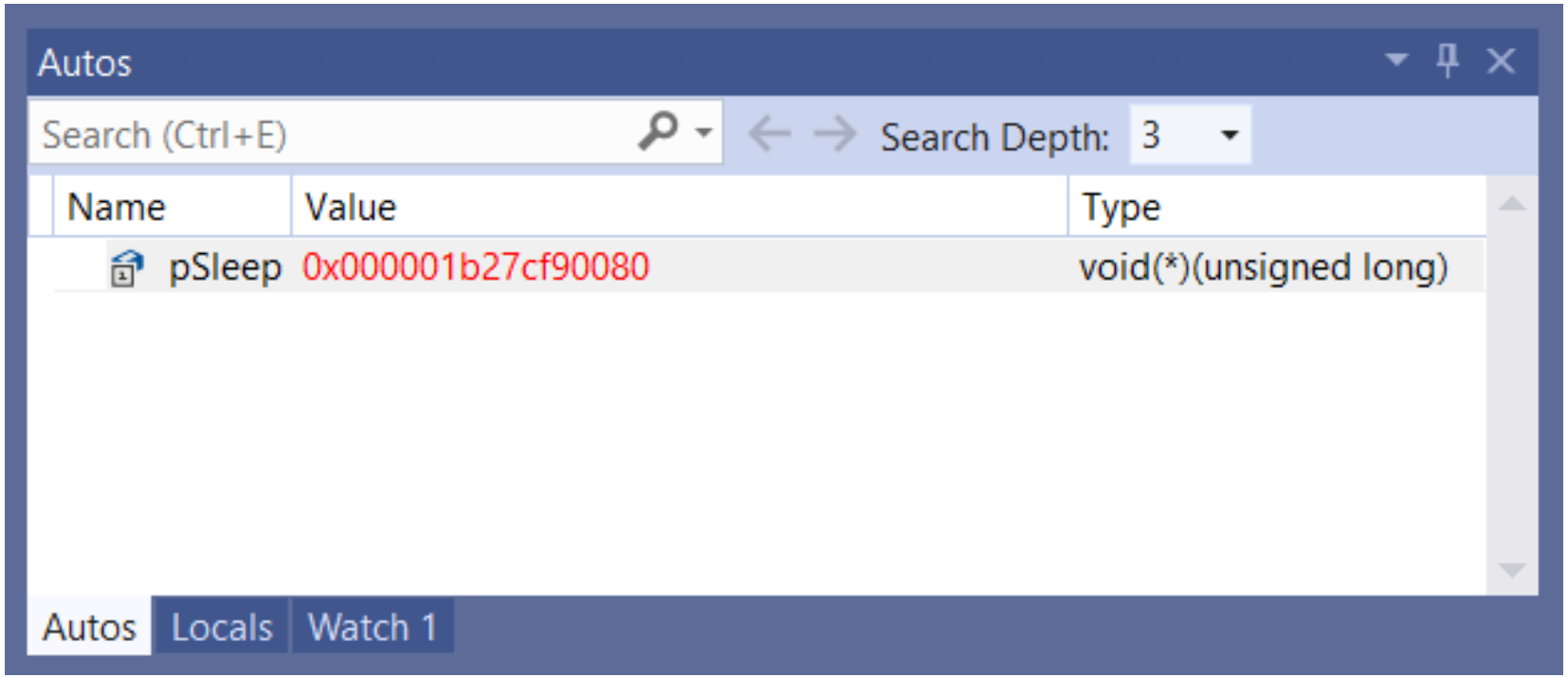
Bitdefender

# Example



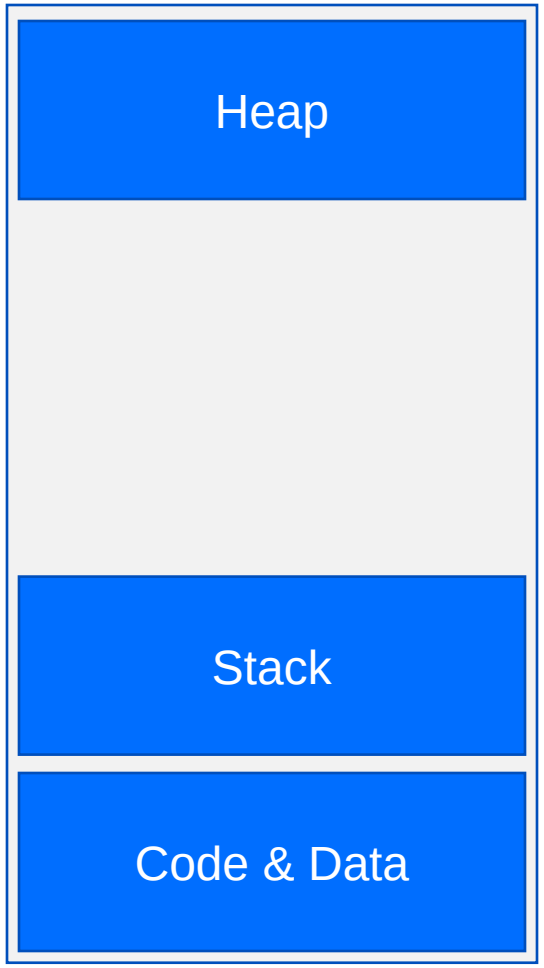
Bitdefender

# Example



Bitdefender

# Finding executable pointers



Bitdefender

# Finding executable pointers

Bitdefender

# Finding executable pointers

Bitdefender

# Finding executable pointers


Read Only Data:

- ❑Import Address Table
- ❑Virtual Function Tables
- ❑Overwriting might be suspicious

Bitdefender

# Finding executable pointers


Read Write Data:

- Arbitrary function pointers

Bitdefender

# Finding executable pointers


```
DSA_DeleteItem
mov     [rsp+arg_0], rbx
push    rdi
sub     rsp, 20h
mov     rax, cs:qword_1403B9A50
mov     ebx, edx
mov     rdi, rcx
cmp     rax, 0FFFFFFFFFFFFFFFFh
jz      short loc_1400104FC
loc_1400104E0:
test    rax, rax
jz      short loc_140010516
mov     edx, ebx
mov     rcx, rdi
call    cs:  guard dispatch icall_fptr
loc_1400104FC:
mov     edx, 146h
lea     rcx, qword_1403B9A50
call    _GetProcFromComCtl32
mov     rax, cs:qword_1403B9A50
jmp     short loc_1400104E0
```

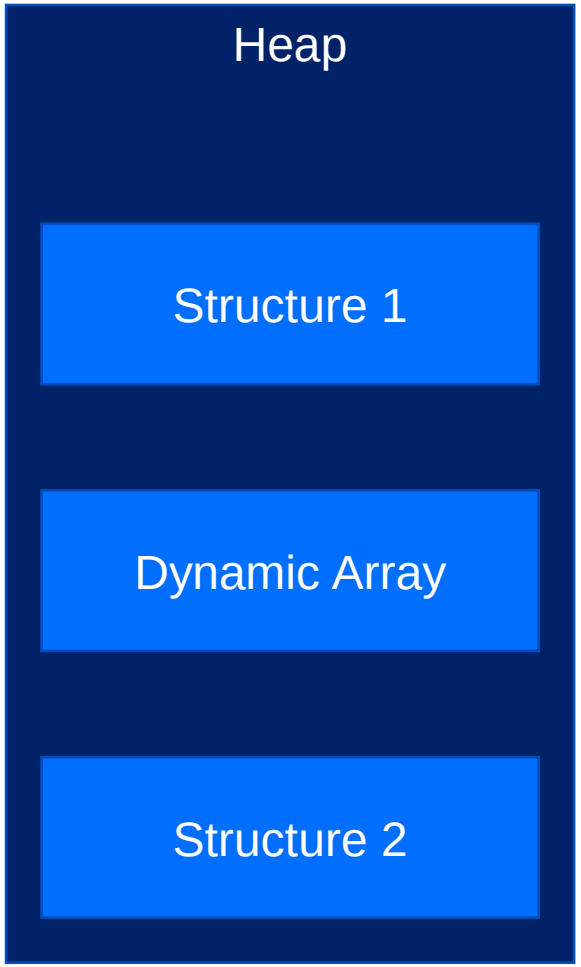
```
_GetProcFromComCtl32
mov     [rsp+arg_0], rbx
mov     [rsp+arg_8], rsi
push    rdi
sub     rsp, 20h
mov     rdi, rcx
xor     ebx, ebx
mov     rcx, cs:g_hinstCC
mov     rsi, rdx
test    rcx, rcx
jz      short loc_14008D2B1
loc_14008D28B:
mov     rdx, rsi
call    cs:__imp_GetProcAddress
nop     dword ptr [rax+rax+00h]
mov     rbx, rax
loc_14008D29D:
mov     [rdi], rbx
mov     rbx, [rsp+28h+arg_0]
mov     rsi, [rsp+28h+arg_8]
add     rsp, 20h
pop     rdi
retn
```

Bitdefender

# Finding executable pointers

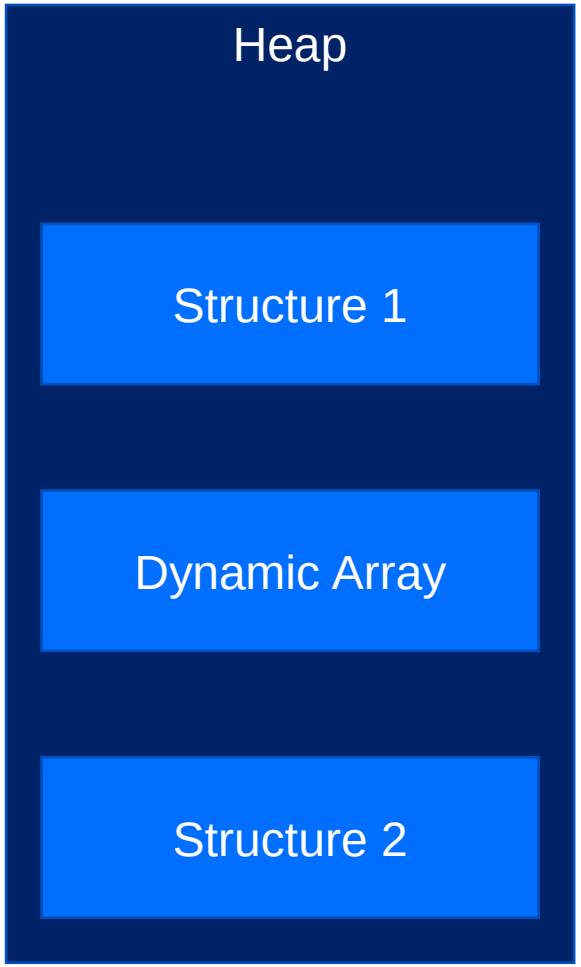
Bitdefender

# Finding executable pointers



Bitdefender

# Finding executable pointers

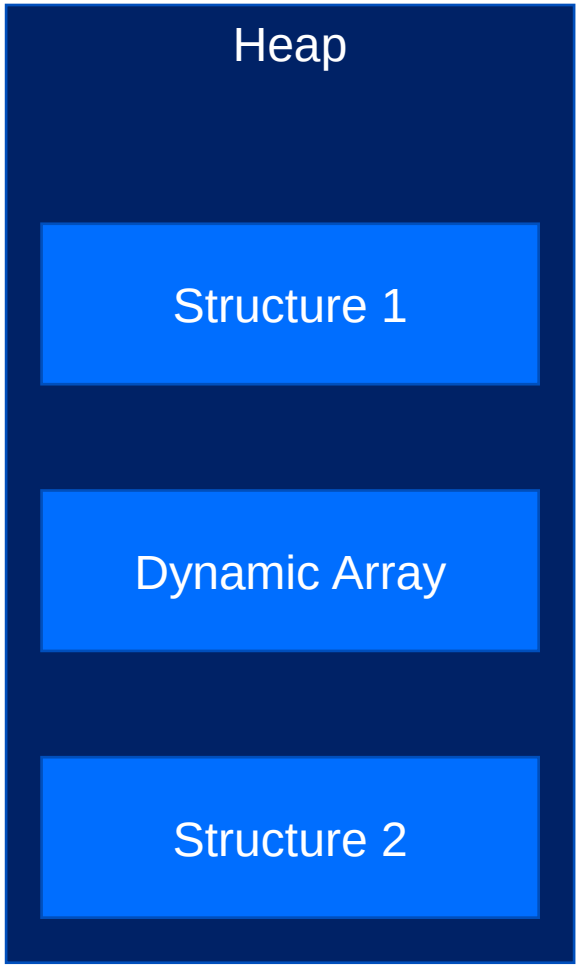



The heap:

- ❑ Arbitrary function pointers
- ❑ Usually contained in various structures

Bitdefender

# Finding executable pointers



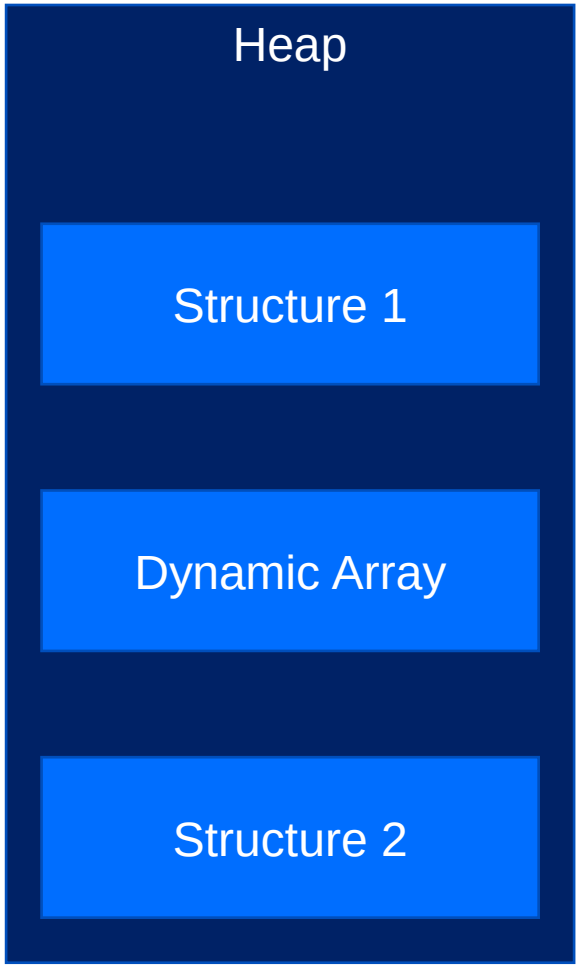

- ❑ TppAlpcpExecuteCallback
- ❑ TppTimerQueueExpiration
- ❑ LdrpWorkCallback

Thread Pool Objects:

- ❑ All processes have a thread pool
- ❑ Represented as structures
- ❑ Multiple work item types
- ❑ Timers, Workers and async Workers (Wait, IO, Alpc)

Bitdefender

# Finding executable pointers




Kernel32!SortCompareString

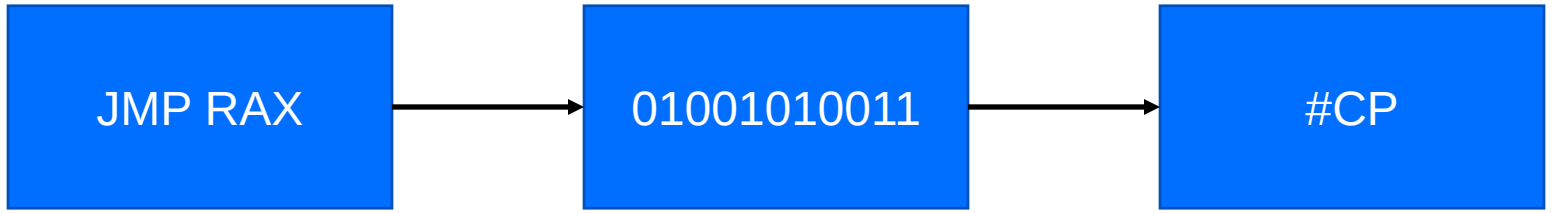
- National Language Support
- Local specific string compare
- Loaded based on registry key

Bitdefender

# Mitigations: CET-IBT

Bitdefender

# Mitigations: CET-IBT



Bitdefender

# Mitigations: CET-IBT

Bitdefender

# Mitigations: CFG

```
while (TRUE) {
        pSleep(1000);
    }
```

Bitdefender

# Mitigations: CFG

```
while (TRUE) {
        pSleep(1000);
    }
```

```
loc_140001028:
mov     ecx, 3E8h
call    rax
mov     rax, cs:pSleep
jmp     short loc_140001028
```

Bitdefender

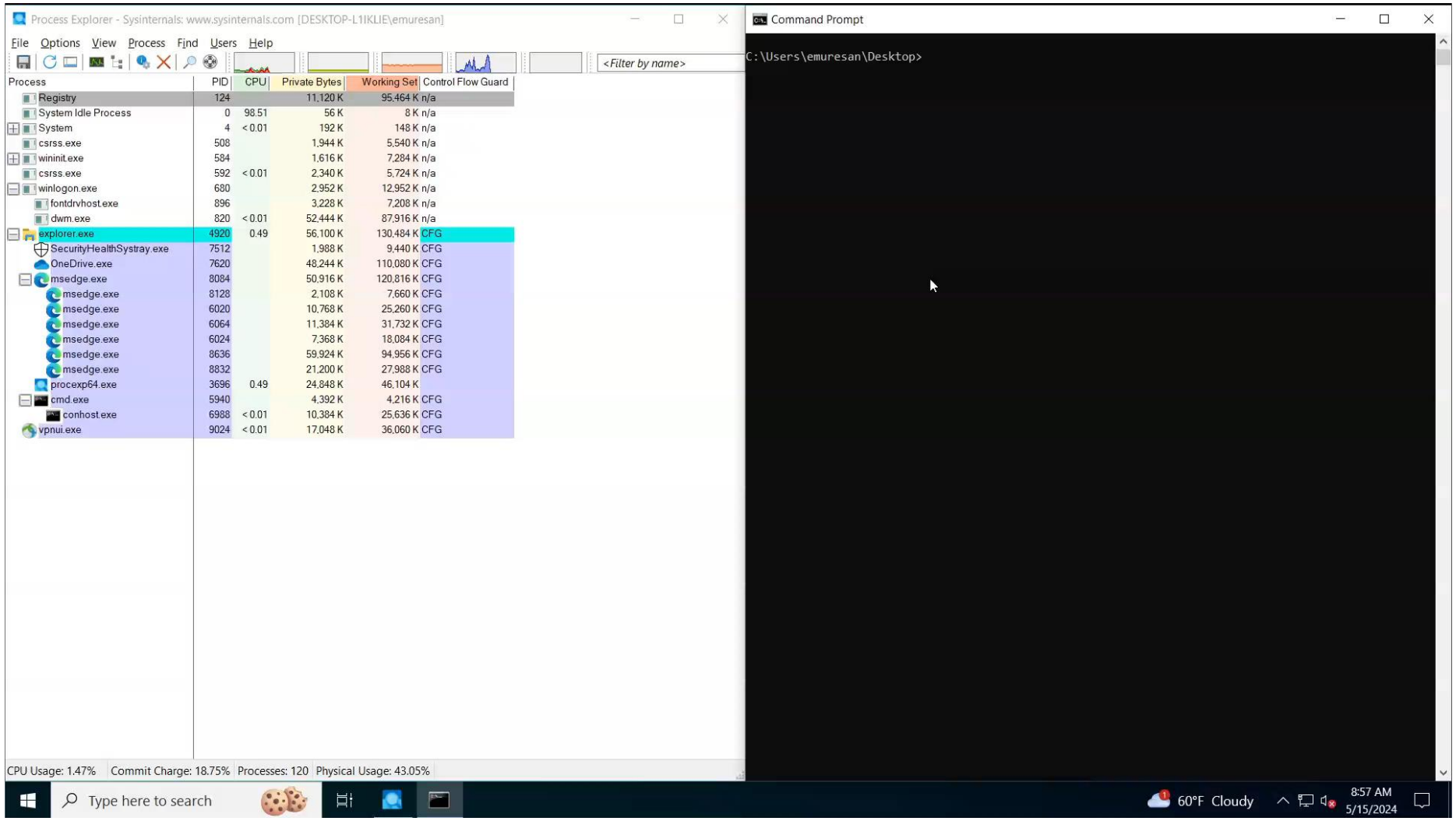
# Mitigations: CFG

```
while (TRUE) {
        pSleep(1000);
    }
```

```
loc_140001030:
mov     ecx, 3E8h
call    cs:__guard_dispatch_icall_fptr
mov     rax, cs:pSleep
jmp     short loc_140001030
```

Bitdefender

# Demo



Bitdefender

# Takeaways

- ❑ Detecting injection attacks by only monitoring execute primitives is necessary but not sufficient.
- ❑ Process Injection allow attackers to evade security solutions, increasing the Mean Time to Detection.
- ❑ Having a comprehensive approach to detecting and preventing injections is mandatory to ensure individuals and organizations remain unharmed via Defense in Depth.
- ❑ C.I.A.P.O. pushes forward the state of the art, demonstrating that attackers might use brute force approaches to overwrite random functions.

# Q&A

Thank you!

Trusted.
**Always.**