StackMoonwalk

A novel approach to stack spoofing on Windows x64

Alessandro Magnosi
Arash Parsa
Athanasios Tserpelis

13/08/2023



WHO WE ARE

Alessandro Magnosi



Athanasios Tserpelis





Arash Parsa



WHO WE ARE

Alessandro Magnosi



bsi.

Principal Security Consultant

Red Teamer, Malware Developer, Source Code Reviewer

PORCHETTA INDUSTRIES

Offensive Security OSS Developer

Author of Inceptor, and several other tools

Synack

Bug Bounty Hunter

Hunting bugs for fun and a little profit

Alessandro Magnosi

KlezVirus

WHO WE ARE

Alessandro Magnosi



Arash Parsa

CYBERARK

Red Team

Red Teamer, Detection Engineer, Threat Hunter

Arash Parsa

waldoirc

waldo-irc

WHO WE ARE





Athanasios Tserpelis



LRQA NETTITUDE

Senior Security Consultant
Red Teamer, Malware Developer, Exploit Developer

Athanasios Tserpelis

trickster012

trickster0

WHO WE ARE

Alessandro Magnosi



Athanasios Tserpelis

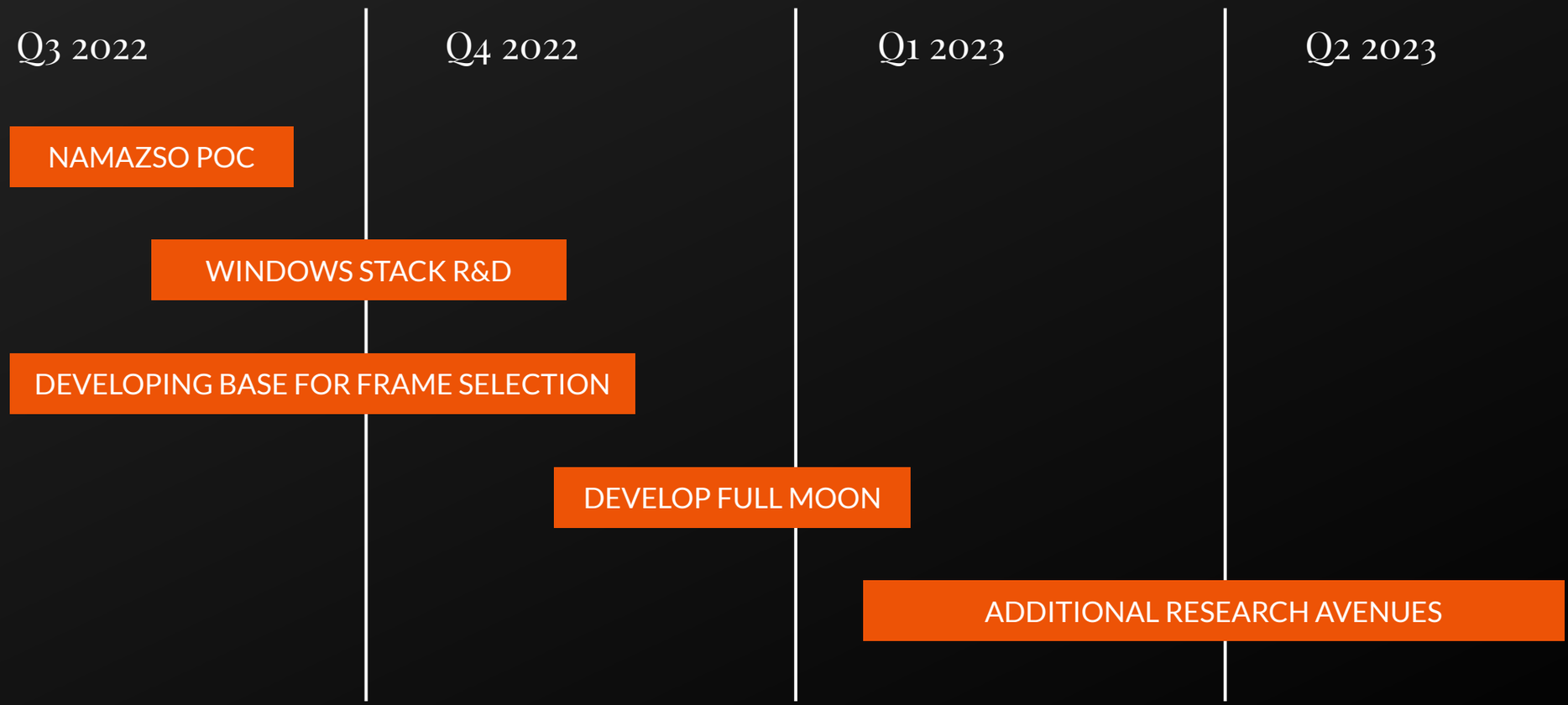




Arash Parsa

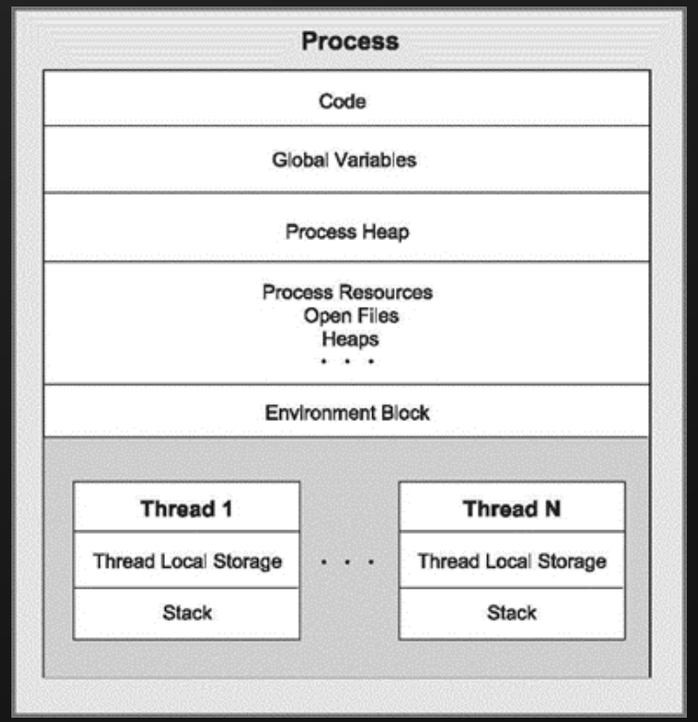


TIMELINE


Q3 2022
Q4 2022
Q1 2023
Q2 2023
NAMAZSO POC
WINDOWS STACK R&D
DEVELOPING BASE FOR FRAME SELECTION
DEVELOP FULL MOON
ADDITIONAL RESEARCH AVENUES

BACKGROUND & PREVIOUS WORK

DEFINE STACK SPOOFING







THREAD CALL STACK

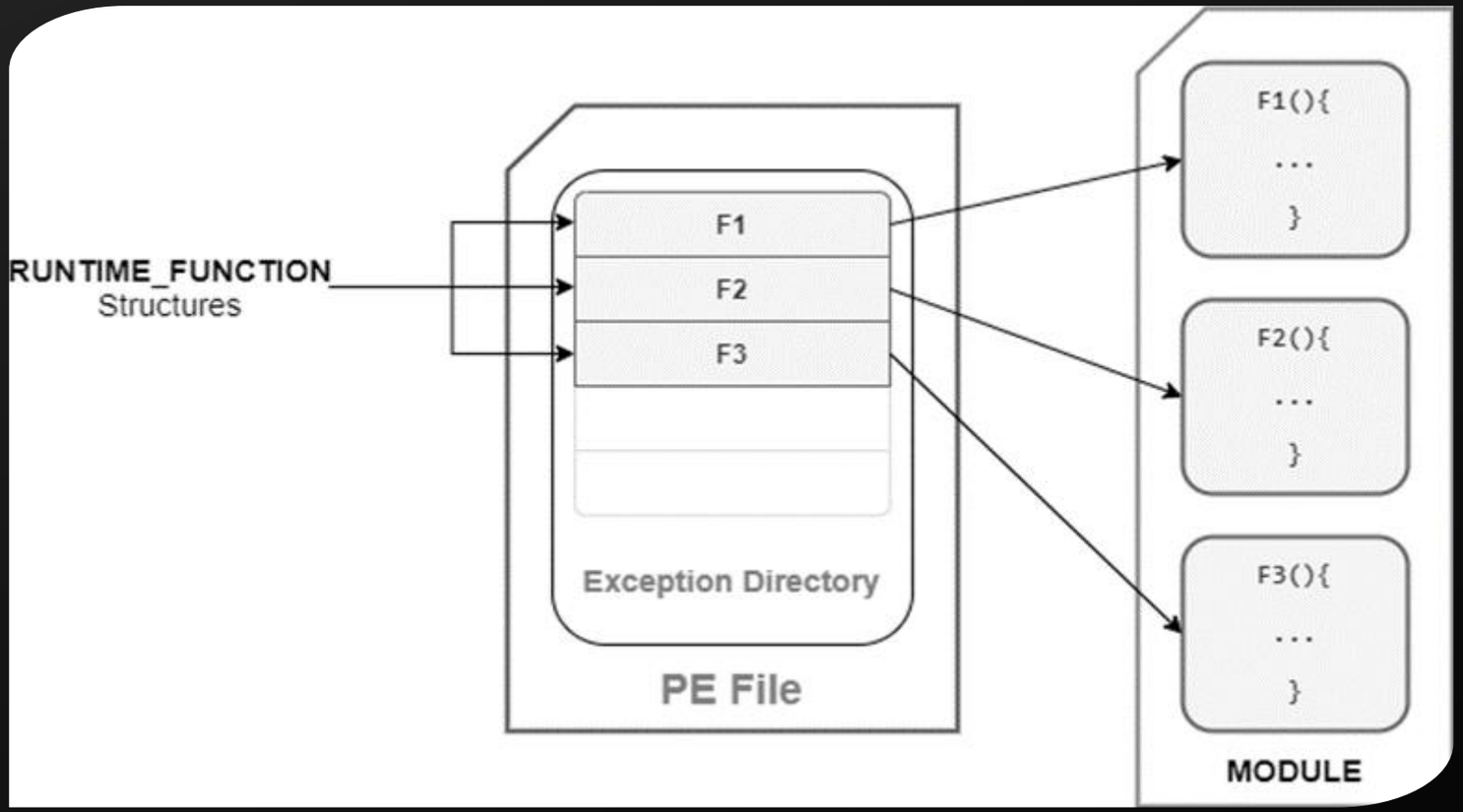
```
00, win32u.dll!NtUserGetMessage+0x14
01, user32.dll!GetMessageW+0x2a
02, notepad++.exe!SetLibraryProperty+0x19ae92
03, notepad++.exe!GetNameSpace+0x105802
04, kernel32.dll!BaseThreadInitThunk+0x1d
05, ntdll.dll!RtlUserThreadStart+0x28
```

```
00, win32u.dll!NtUserGetMessage+0x14
01, user32.dll!GetMessageW+0x2a
02, Notepad.exe+0x734c6
03, Notepad.exe+0x18236
04, Notepad.exe+0x40e70
05, Notepad.exe+0x7b55e
06, kernel32.dll!BaseThreadInitThunk+0x1d
07, ntdll.dll!RtlUserThreadStart+0x28
```

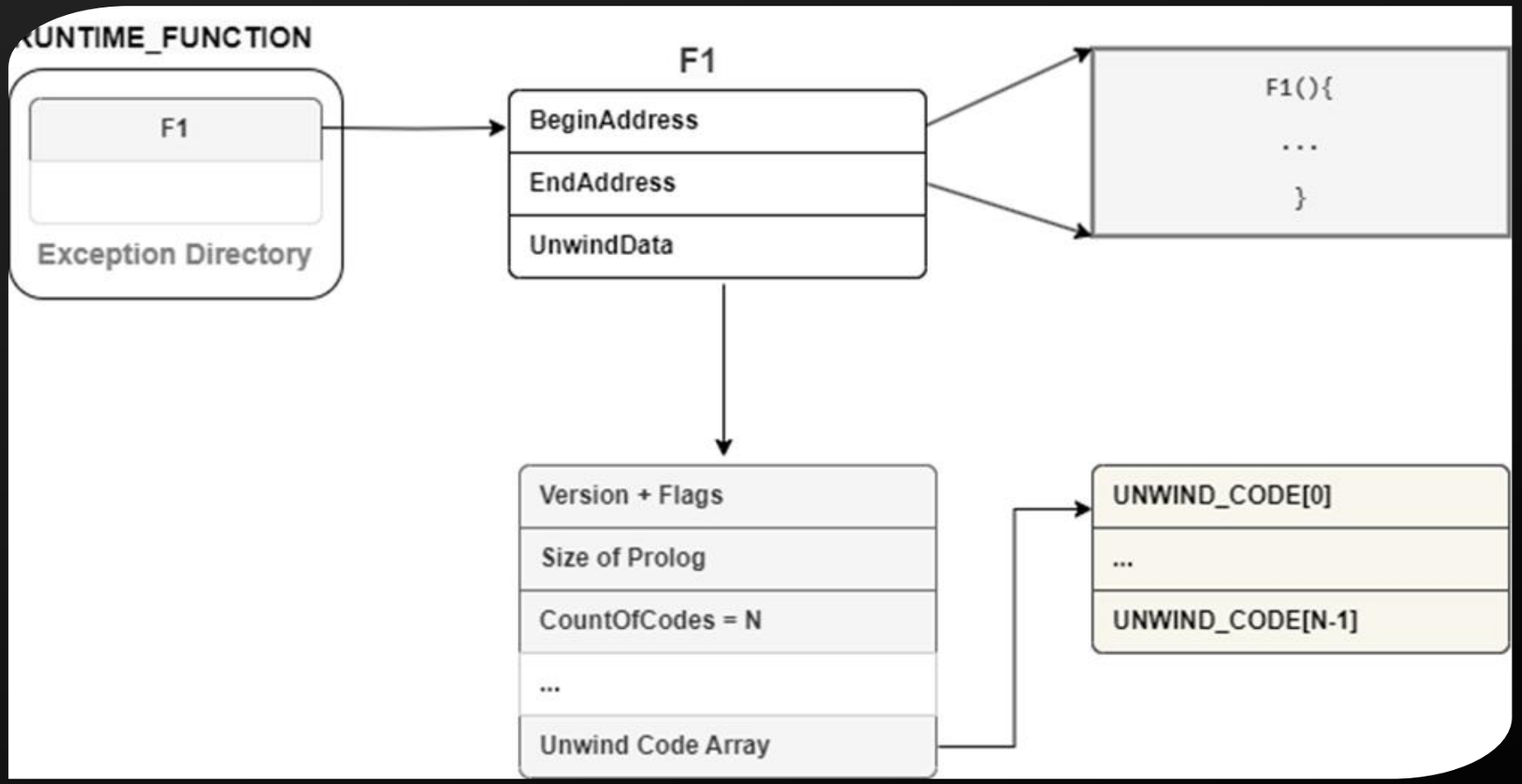
```
00, ntdll.dll!NtDeviceIoControlFile+0x14
01, KernelBase.dll!GetConsoleScreenBufferInfoEx+0x229
02, KernelBase.dll!ReadConsoleW+0x1c5
03, KernelBase.dll!ReadConsoleW+0x1e
04, cmd.exe+0x2bf87
05, cmd.exe+0x13b11
06, cmd.exe+0x1305a
07, cmd.exe+0x12d66
08, cmd.exe+0x12ad2
09, cmd.exe+0x25b2d
10, cmd.exe+0x1f876
11, kernel32.dll!BaseThreadInitThunk+0x1d
12, ntdll.dll!RtlUserThreadStart+0x28
```

```
00, ntdll.dll!NtWaitForSingleObject+0x14
01, KernelBase.dll!DeviceIoControl+0x86
02, kernel32.dll!DeviceIoControl+0x81
03, conhost.exe+0x16490
04, conhost.exe+0x94a6
05, kernel32.dll!BaseThreadInitThunk+0x1d
06, ntdll.dll!RtlUserThreadStart+0x28
```

UNWINDING

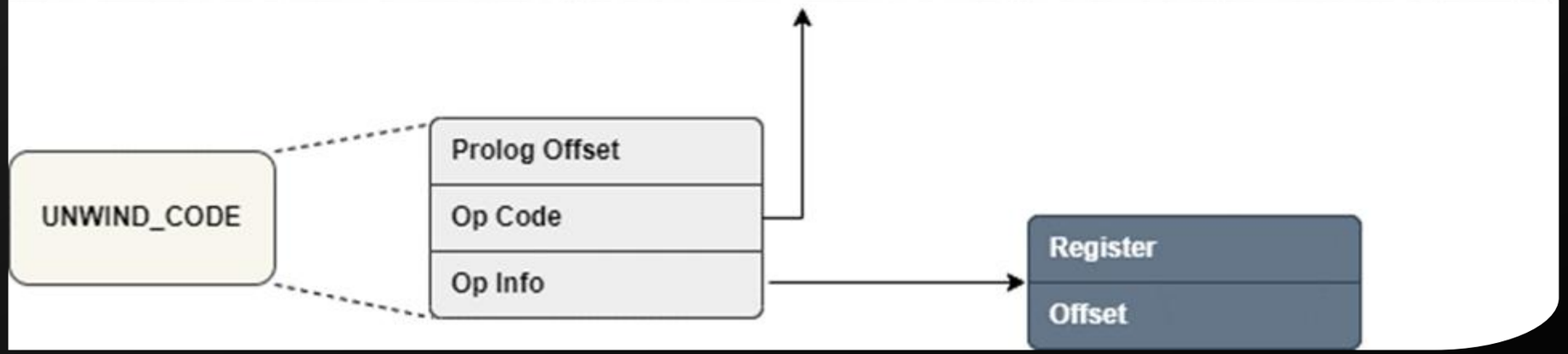

RUNTIME_FUNCTION
Structures
F1
F2
F3
Exception Directory
PE File
F1(){
...
}
F2(){
...
}
F3(){
...
}
MODULE

UNWINDING

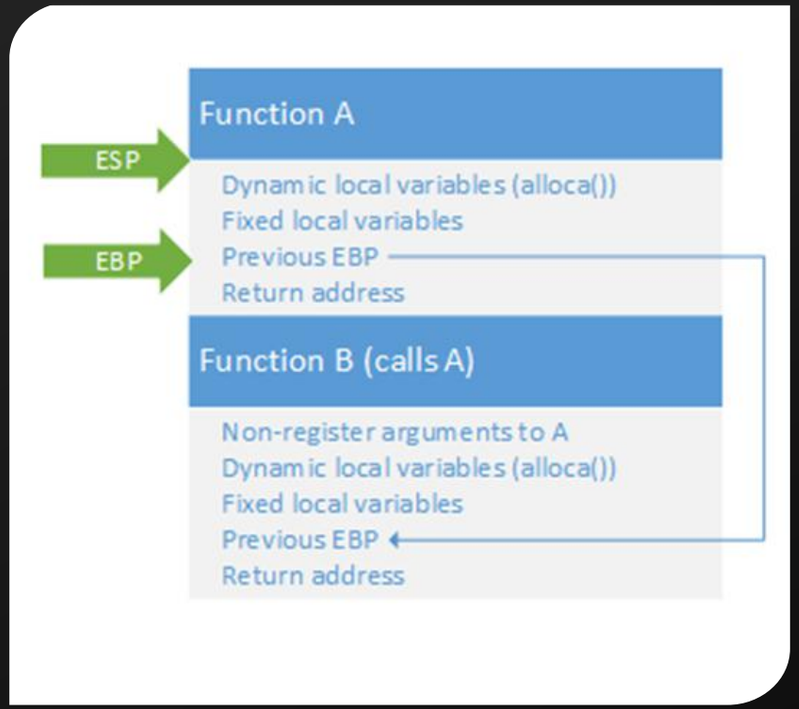

UNTIME_FUNCTION
F1
Exception Directory
F1
BeginAddress
EndAddress
UnwindData
F1(){
...
}
Version + Flags
Size of Prolog
CountOfCodes = N
...
Unwind Code Array
UNWIND_CODE[0]
...
UNWIND_CODE[N-1]

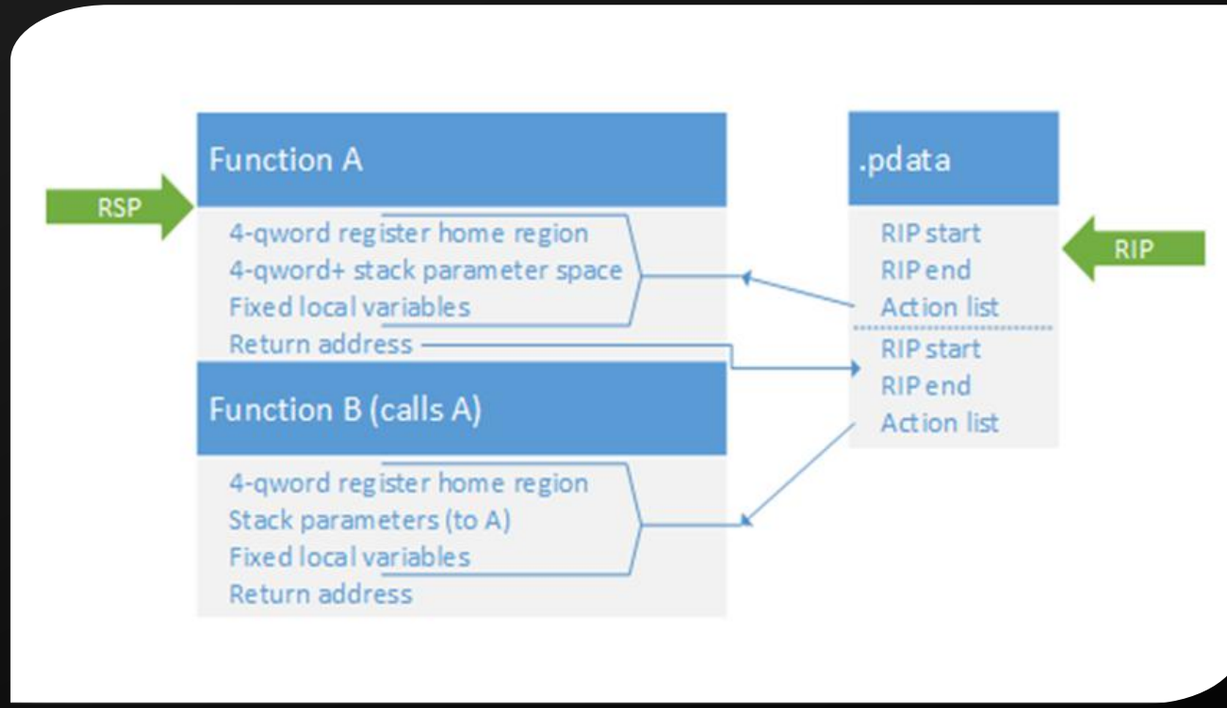
UNWINDING

Impact Frame Size	*Do not Impact Frame Size*
UWOP_PUSH_NONVOL	UWOP_SET_FPREG
UWOP_ALLOC_SMALL	UWOP_SAVE_NONVOL
UWOP_ALLOC_LARGE	UWOP_SAVE_NONVOL_FAR
UWOP_PUSH_MACHFRAME	UWOP_SAVE_XMM128_FAR
	UWOP_SAVE_XMM128



WINDOWS CALLING CONVENTION





https://www.leviathansecurity.com/media/use-of-windows-exception-handling-metadata

CALL-STACK ANALYSIS

In-Memory Execution

When a particular API or system call is executed, Call Stack analysis can be utilised to trace back the caller frame and verify if it can be resolved to a module on disk.

In-Direct System Calls

Although rarely used for this reason, analyzing the call stack can be used to identify situation whereby a system call has been invoked directly via an Nt function or by using an indirect jump.

IN-MEMORY EXECUTION

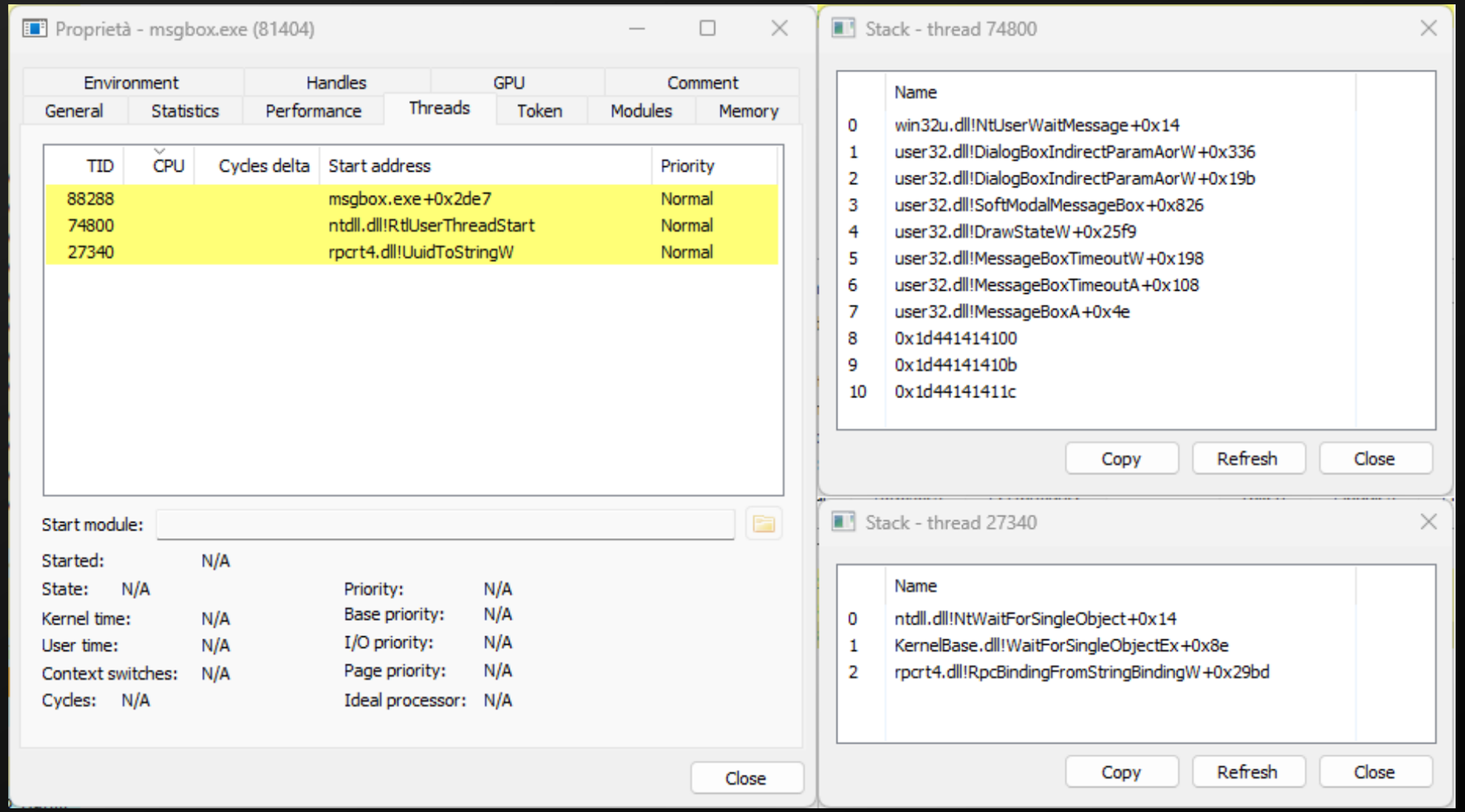
Normal Call Stack

```
0, win32u.dll!NtUserWaitMessage+0x12
1, user32.dll!DialogBoxIndirectParamAorW+0x336
2, user32.dll!DialogBoxIndirectParamAorW+0x19b
3, user32.dll!SoftModalMessageBox+0x826
4, user32.dll!DrawStateW+0x25f9
5, user32.dll!MessageBoxTimeoutW+0x198
6, user32.dll!MessageBoxTimeoutA+0x108
7, user32.dll!MessageBoxA+0x4e
8, AbsentMoon.exe!main+0xed
9, AbsentMoon.exe!__scrt_common_main_seh+0x10c
10, kernel32.dll!BaseThreadInitThunk+0x1d
11, ntdll.dll!RtlUserThreadStart+0x28
```

In-Memory Code Call Stack

```
0, win32u.dll!NtUserWaitMessage+0x12
1, user32.dll!DialogBoxIndirectParamAorW+0x336
2, user32.dll!DialogBoxIndirectParamAorW+0x19b
3, user32.dll!SoftModalMessageBox+0x826
4, user32.dll!DrawStateW+0x25f9
5, user32.dll!MessageBoxTimeoutW+0x198
6, user32.dll!MessageBoxTimeoutA+0x108
7, user32.dll!MessageBoxA+0x4e
8, 0x0000000000000000
```

IN-MEMORY EXECUTION - STOMPING



INDIRECT SYSCALL EXECUTION

High Level API Call Stack

```
0, ntdll.dll!NtAllocateVirtualMemory+0x13
1, KernelBase.dll!VirtualAllocExNuma+0x5c
2, KernelBase.dll!VirtualAllocEx+0x16
3, AbsentMoon.exe!main+0x5a
4, AbsentMoon.exe!__scrt_common_main_seh+0x10c
5, kernel32.dll!BaseThreadInitThunk+0x1d
6, ntdll.dll!RtlUserThreadStart+0x28
```

Indirect Syscall Call Stack

```
0, ntdll.dll!NtAllocateVirtualMemory+0x13


1, AbsentMoon.exe!main+0xc4
2, AbsentMoon.exe!__scrt_common_main_seh+0x10c
3, kernel32.dll!BaseThreadInitThunk+0x1d
4, ntdll.dll!RtlUserThreadStart+0x28
```

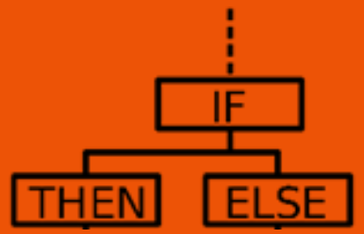
WHEN TO DETECT

Periodic



Every
period of T seconds all
threads can be scanned
and their call stack analysed

Conditional



A security tool might scan
all threads in a WAIT state
to check for sleeping
implants

Hooking



Every
time a specific API or
System Call gets called, the
call stack can be analysed

PREVIOUS RESEARCH

1

Stack Truncation

This is an extension of return address spoofing, which ensure to zero out the return address of the caller frame

2

Stack Crafting

Craft the thread call stack artificially, to mimic other legitimate threads when invoking a specific API

3

Stack Cloning

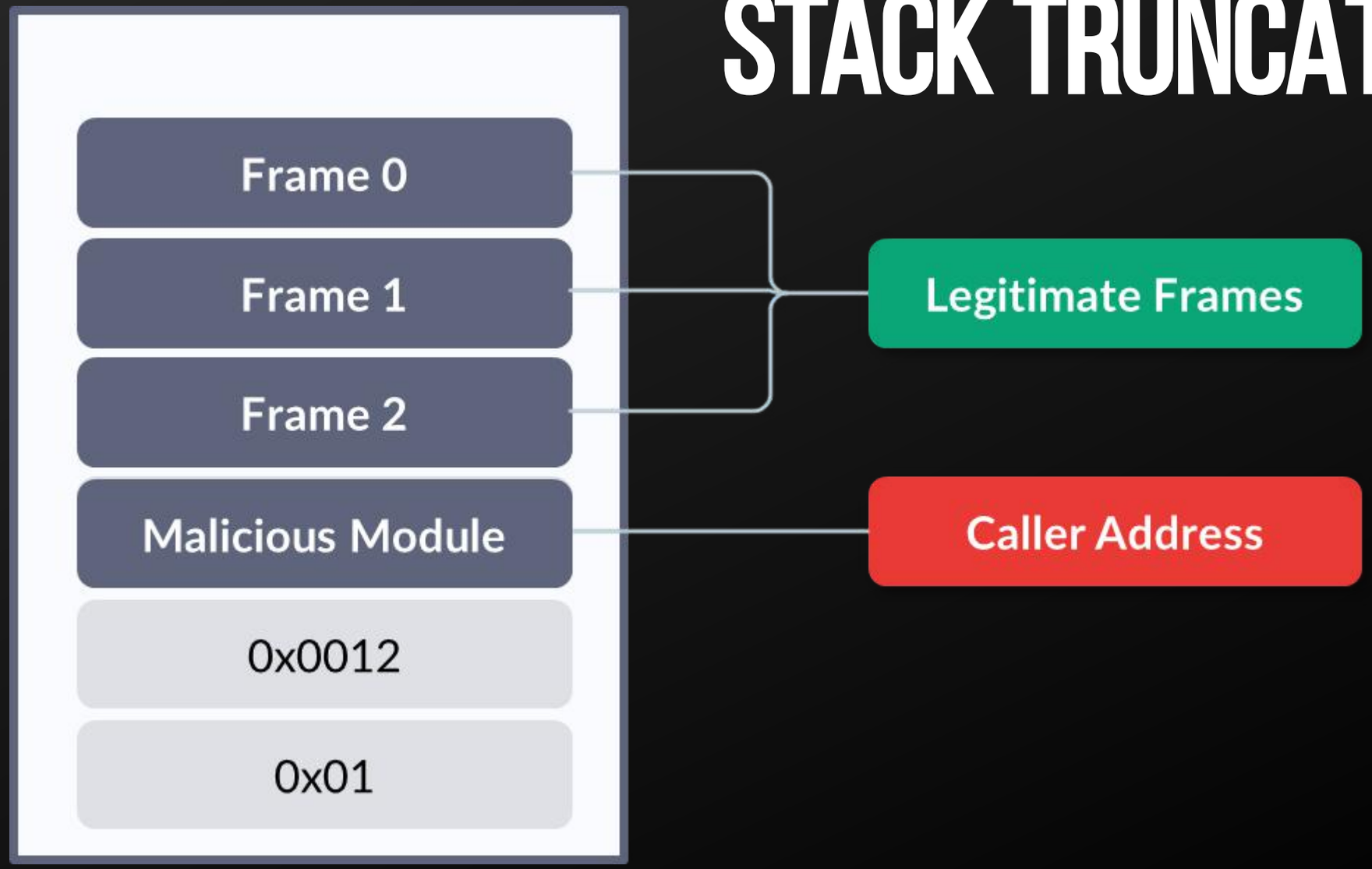
Uses an external mechanism (timers, APC queues) to hide the thread stack of the in-memory injected code by cloning a legitimate thread stack

4

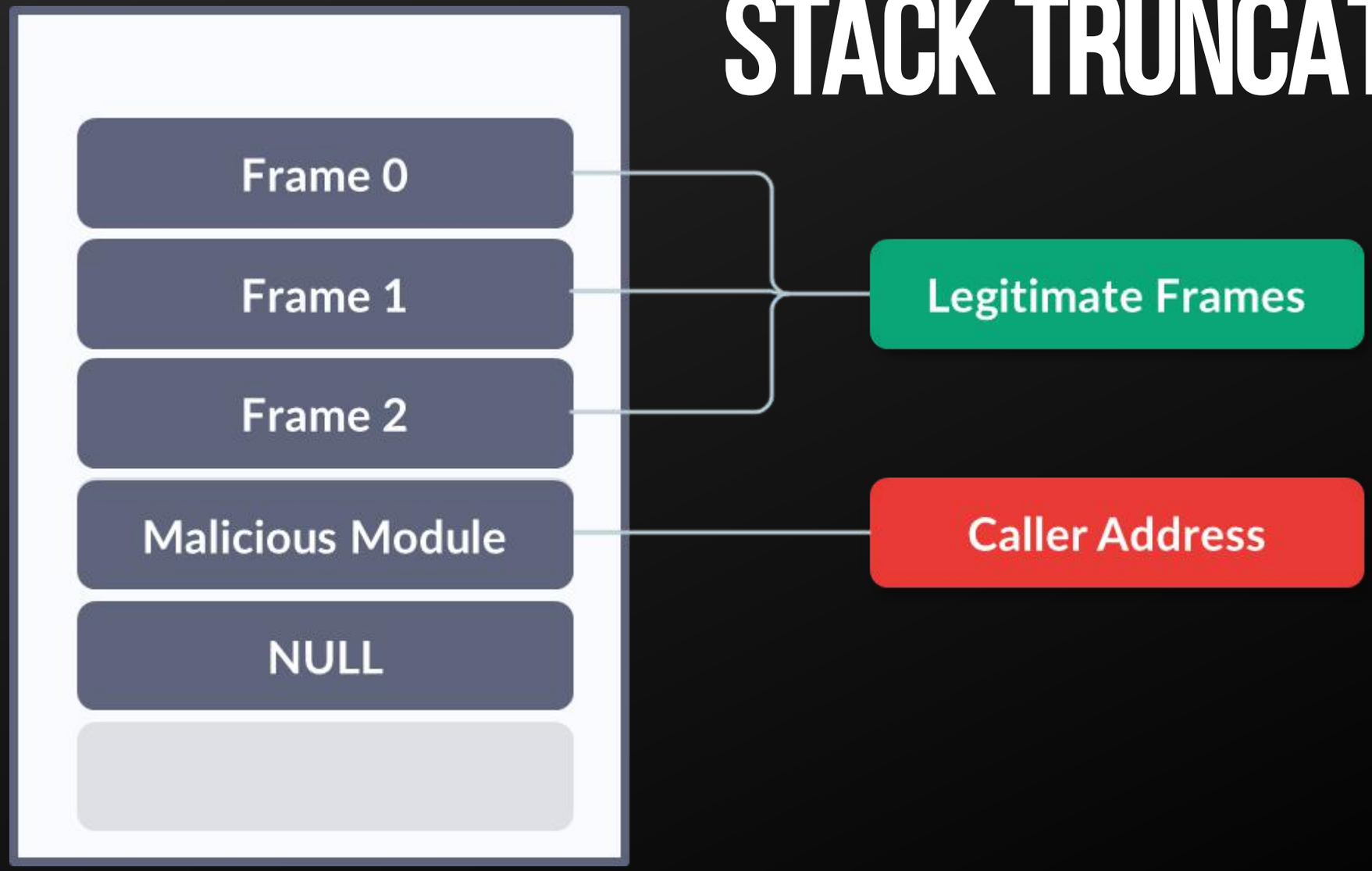
Stack Hiding

Uses Fibers to hide the thread stack of the in-memory injected code

STACK TRUNCATION



STACK TRUNCATION



STACK TRUNCATION

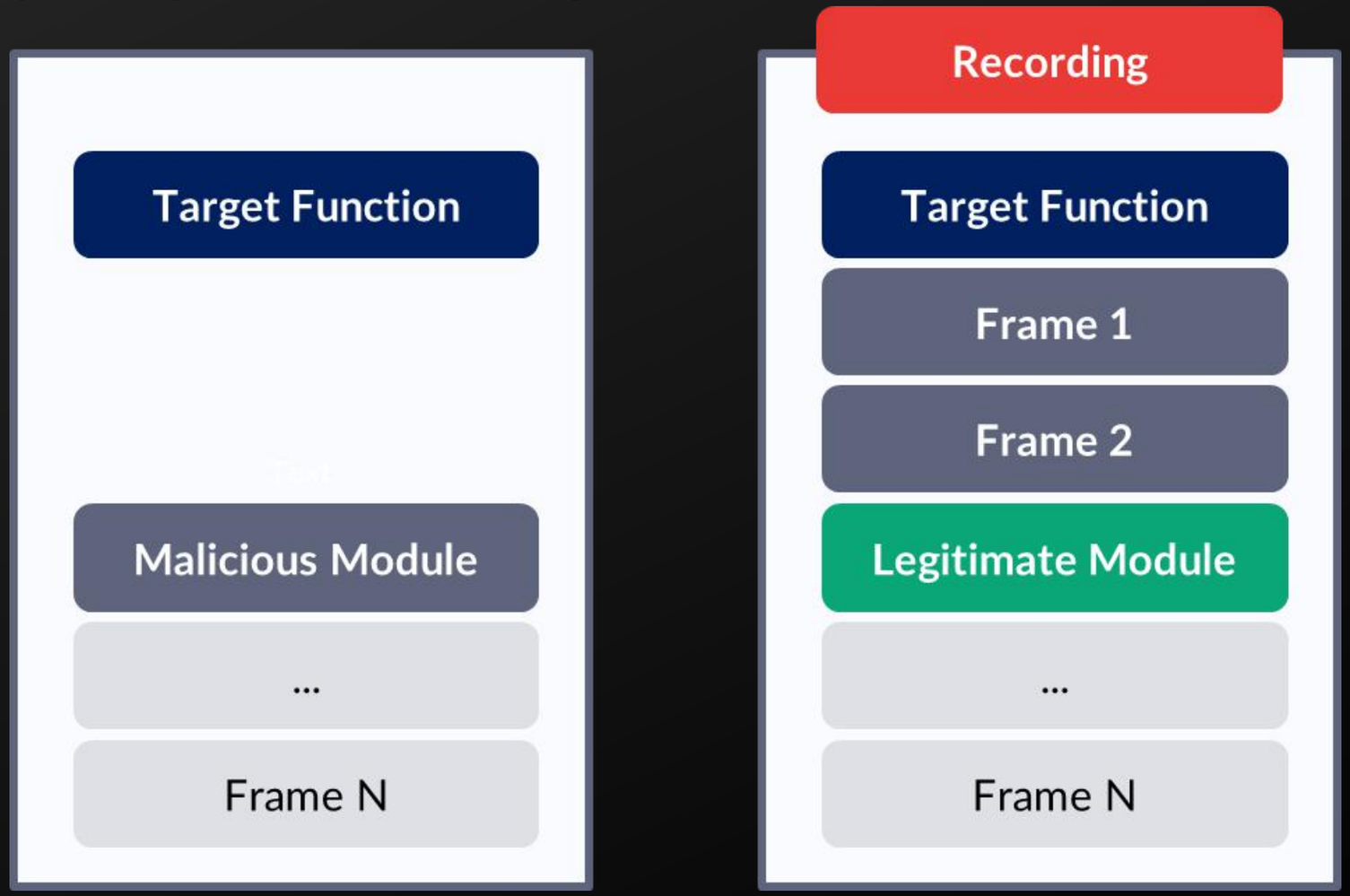
Stack - thread 34356

#	Name	Stack address	Return address	Frame address
0	ntoskrnl.exe!KiDeliverApc+0x1b0			
1	ntoskrnl.exe!KiSwapThread+0x827			
2	ntoskrnl.exe!KiCommitThreadWait+0x14f			
3	ntoskrnl.exe!KeDelayExecutionThread+0x122			
4	ntoskrnl.exe!NtDelayExecution+0x5f			
5	ntoskrnl.exe!KiSystemServiceCopyEnd+0x25			
6	ntdll.dll!NtDelayExecution+0x14	0x88da5ffa98	0x7ffeb65795be	0x88da5ffa90
7	KernelBase.dll!SleepEx+0x9e	0x88da5ffaa0	0x22d6bd5bd51	0x88da5ffb30
8	0x22d6bd5bd51	0x88da5ffb40	0x1388	0x88da5ffb38
9	0x1388	0x88da5ffb48	0x22d00000000	0x88da5ffb40
10	0x22d00000000	0x88da5ffb50	0x1b0001c00000bb	0x88da5ffb48
11	0x1b0001c00000bb	0x88da5ffb58		0x88da5ffb50

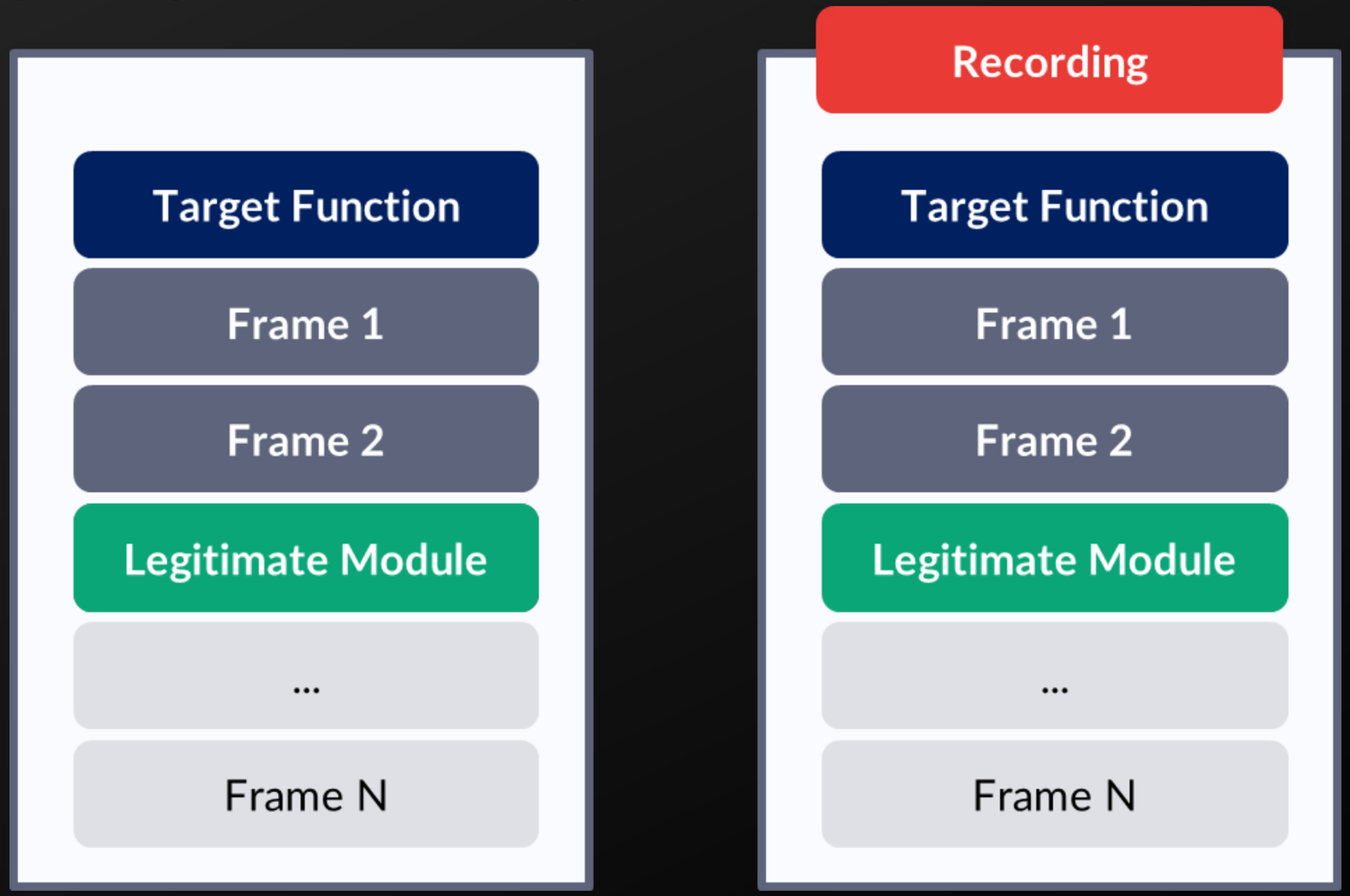
Stack - thread 45956

#	Name	Stack address	Frame address	Return address
0	ntoskrnl.exe!KiDeliverApc+0x1b0			
1	ntoskrnl.exe!KiSwapThread+0x827			
2	ntoskrnl.exe!KiCommitThreadWait+0x14f			
3	ntoskrnl.exe!KeDelayExecutionThread+0x122			
4	ntoskrnl.exe!NtDelayExecution+0x5f			
5	ntoskrnl.exe!KiSystemServiceCopyEnd+0x25			
6	ntdll.dll!NtDelayExecution+0x14	0x3211ff4d8	0x3211ff4d0	0x7ffeb65795be
7	KernelBase.dll!SleepEx+0x9e	0x3211ff4e0	0x3211ff570	0x7ff79a49125c
8	ThreadStackSpoofer.exe!MySleep+0x5c	0x3211ff580	0x3211ff5d0	

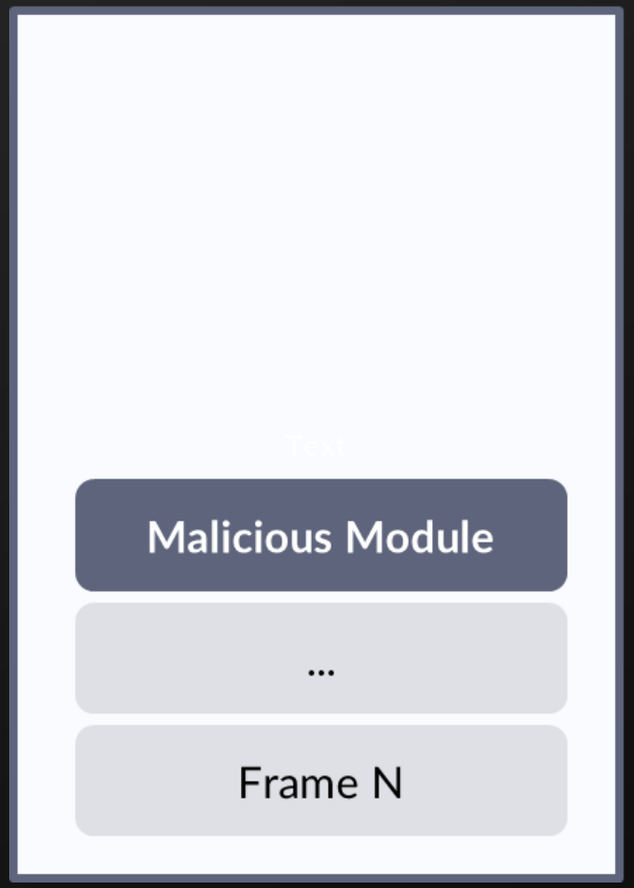
STACK CRAFTING

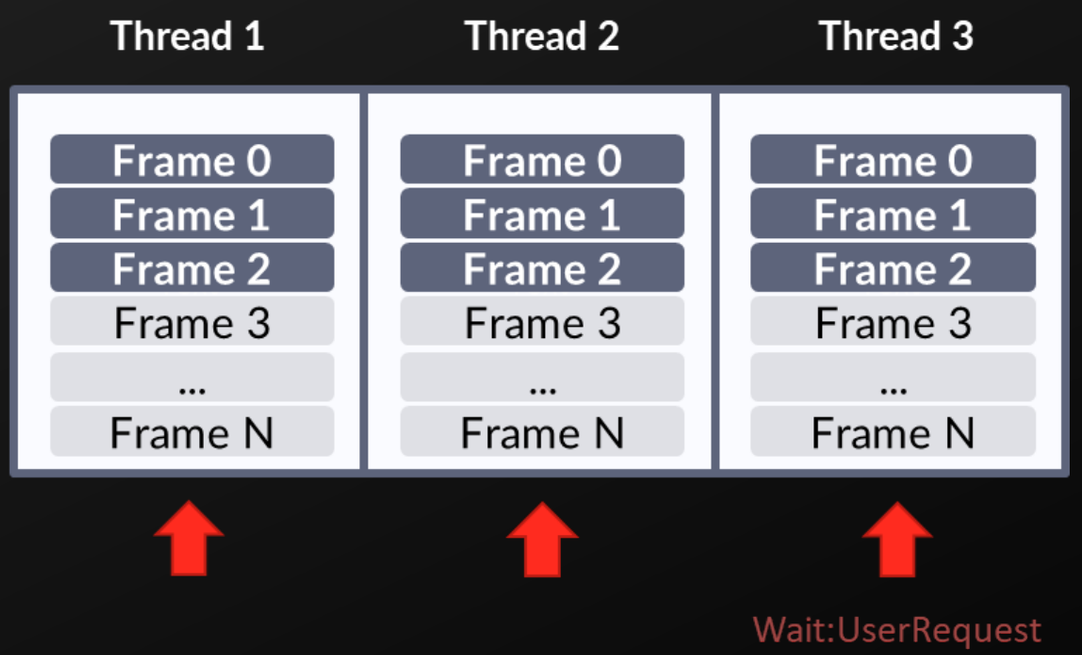


STACK CRAFTING

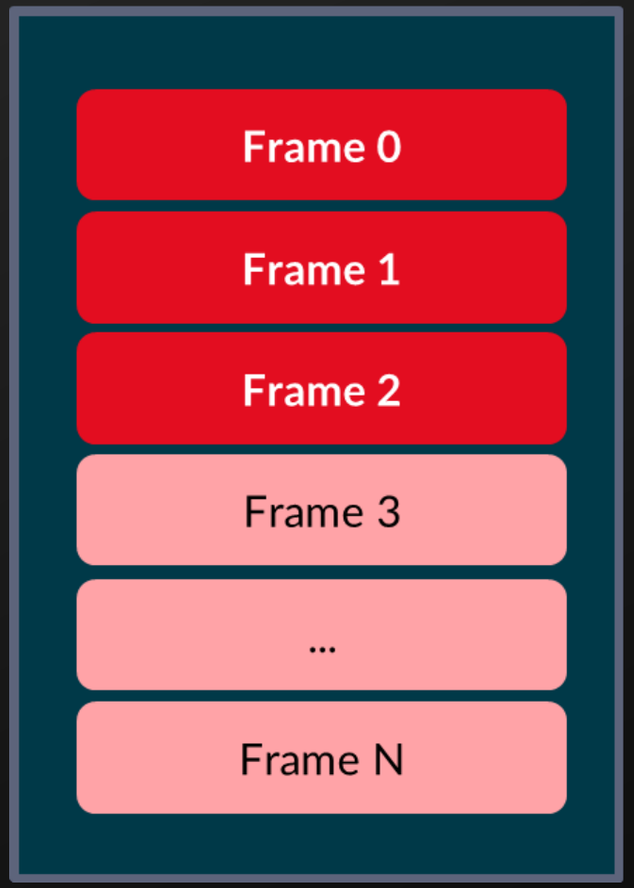


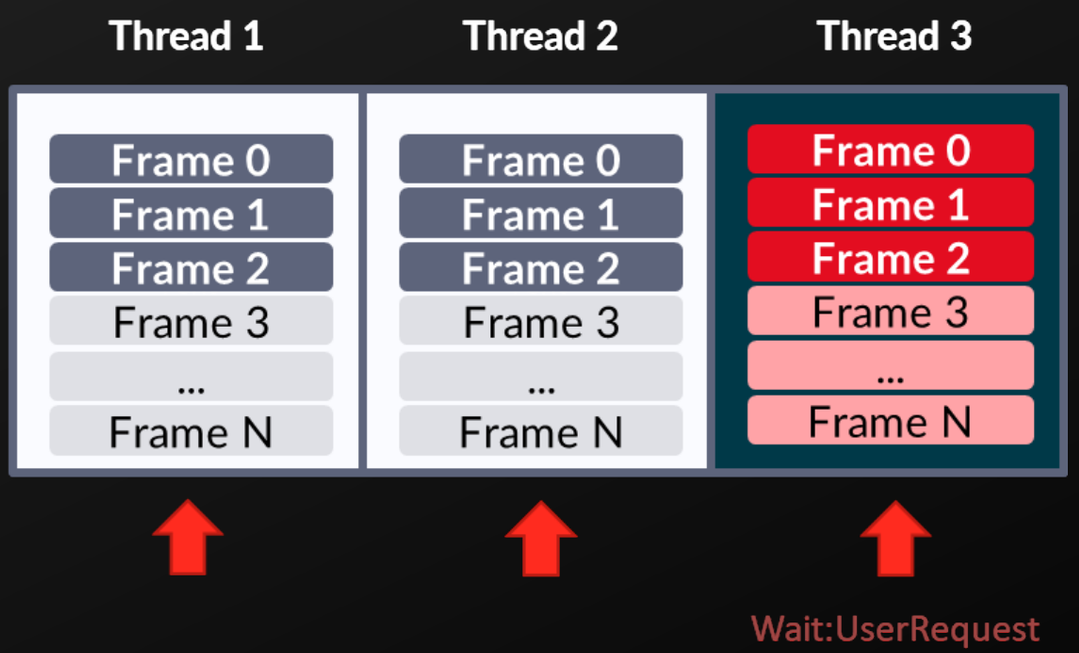
STACK CLONING





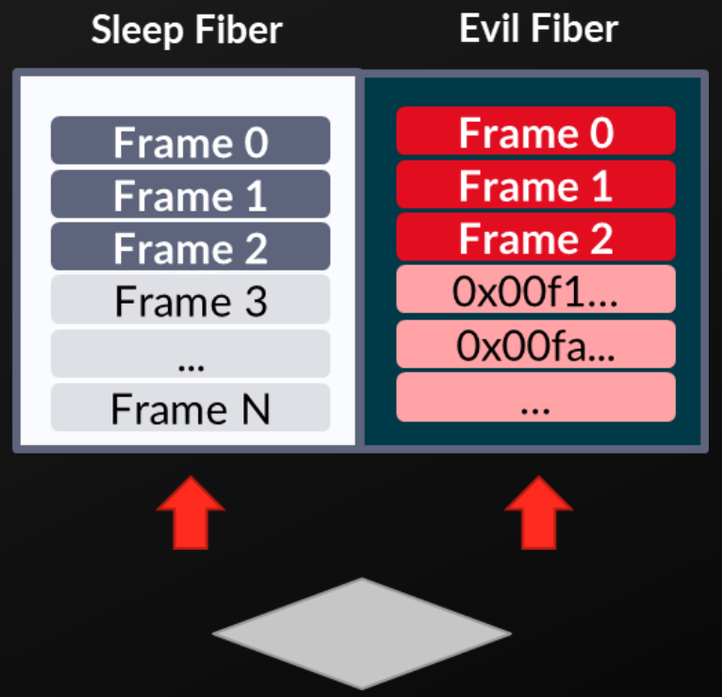
STACK CLONING





STACK HIDING





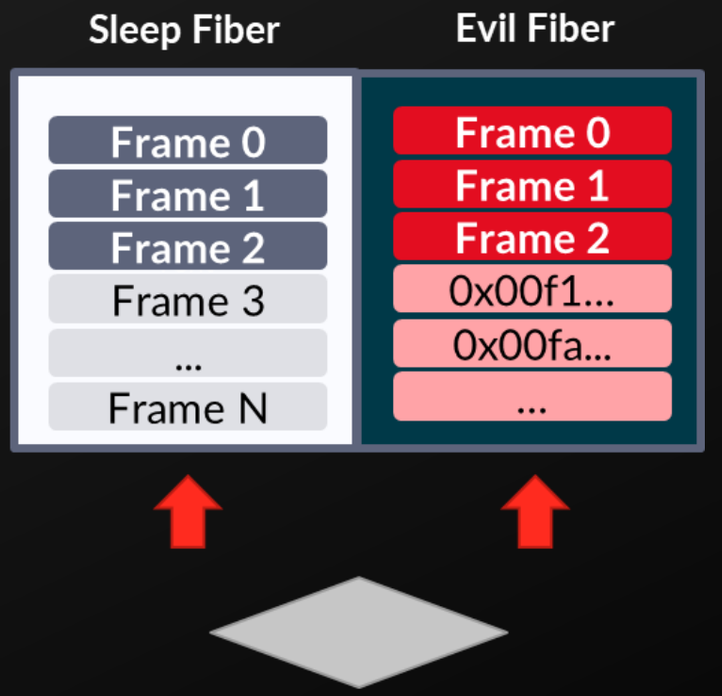
SwitchToFiber(x)

ConvertThreadToFiber(x)

CreateFiber(x, y, z)

STACK HIDING



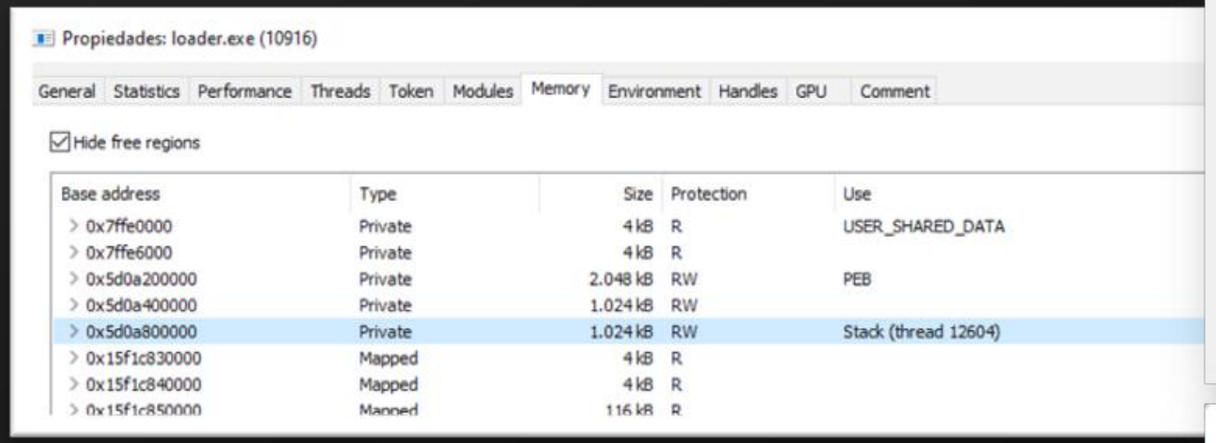


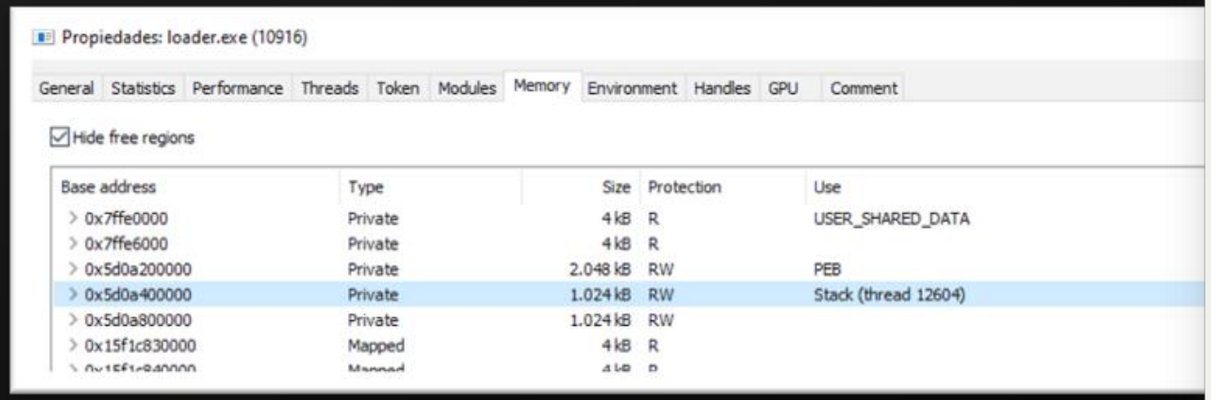
SwitchToFiber(x)

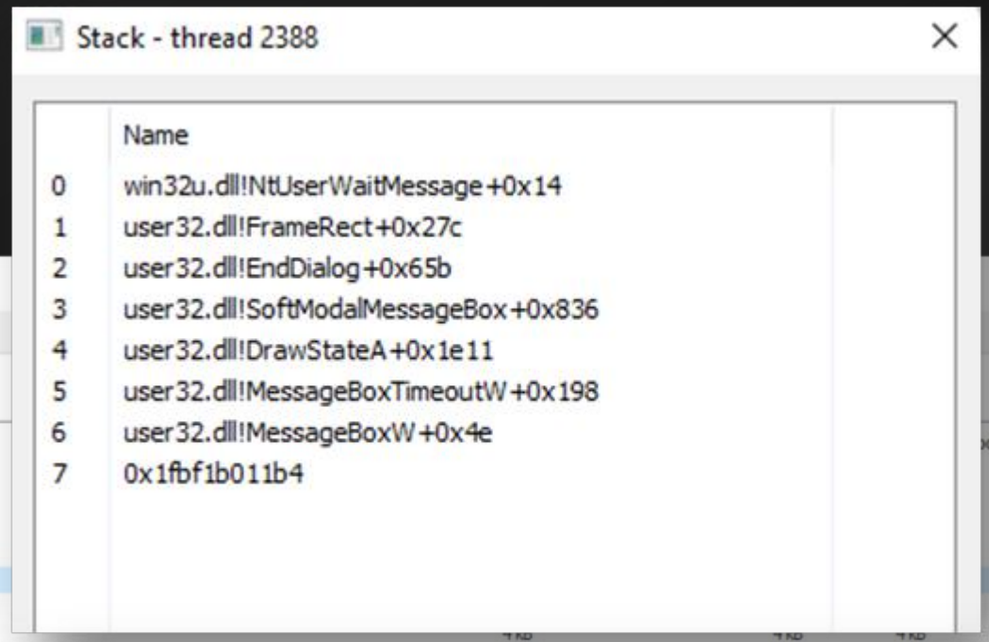
ConvertThreadToFiber(x)

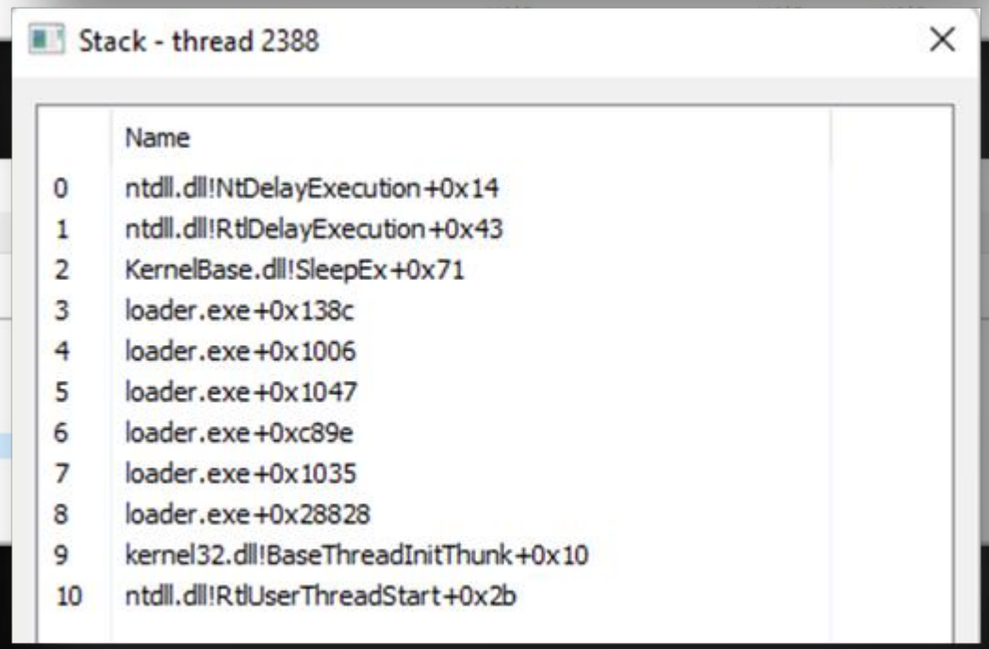
CreateFiber(x, y, z)

STACK HIDING









OUR CONTRIBUTION

STACK MOONWWALKING

1 Hide Caller

This technique was initially developed to conceal the main caller module from the call stack, preseriving its unwindability when executing in-memory code

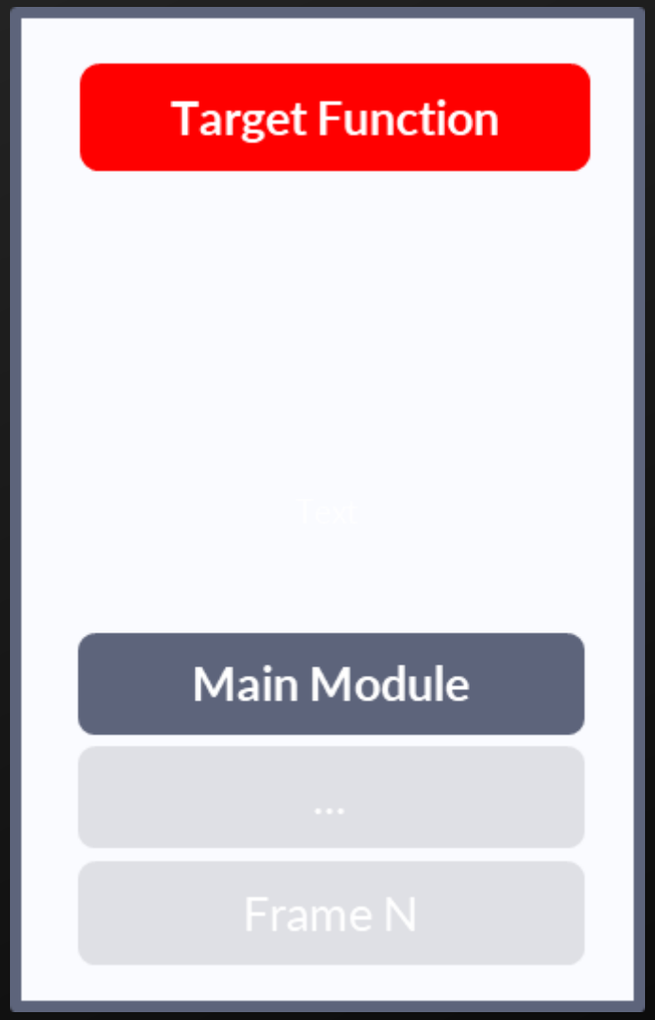
2 Obfuscate Stack

Stack Moonwalk obfuscates the stack, making it difficult to understand how a specific API/System Call was invoked

3 Auto Restore

The technique does not use any external mechanism to restore the stack on return, it does use a ROP paradigm instead

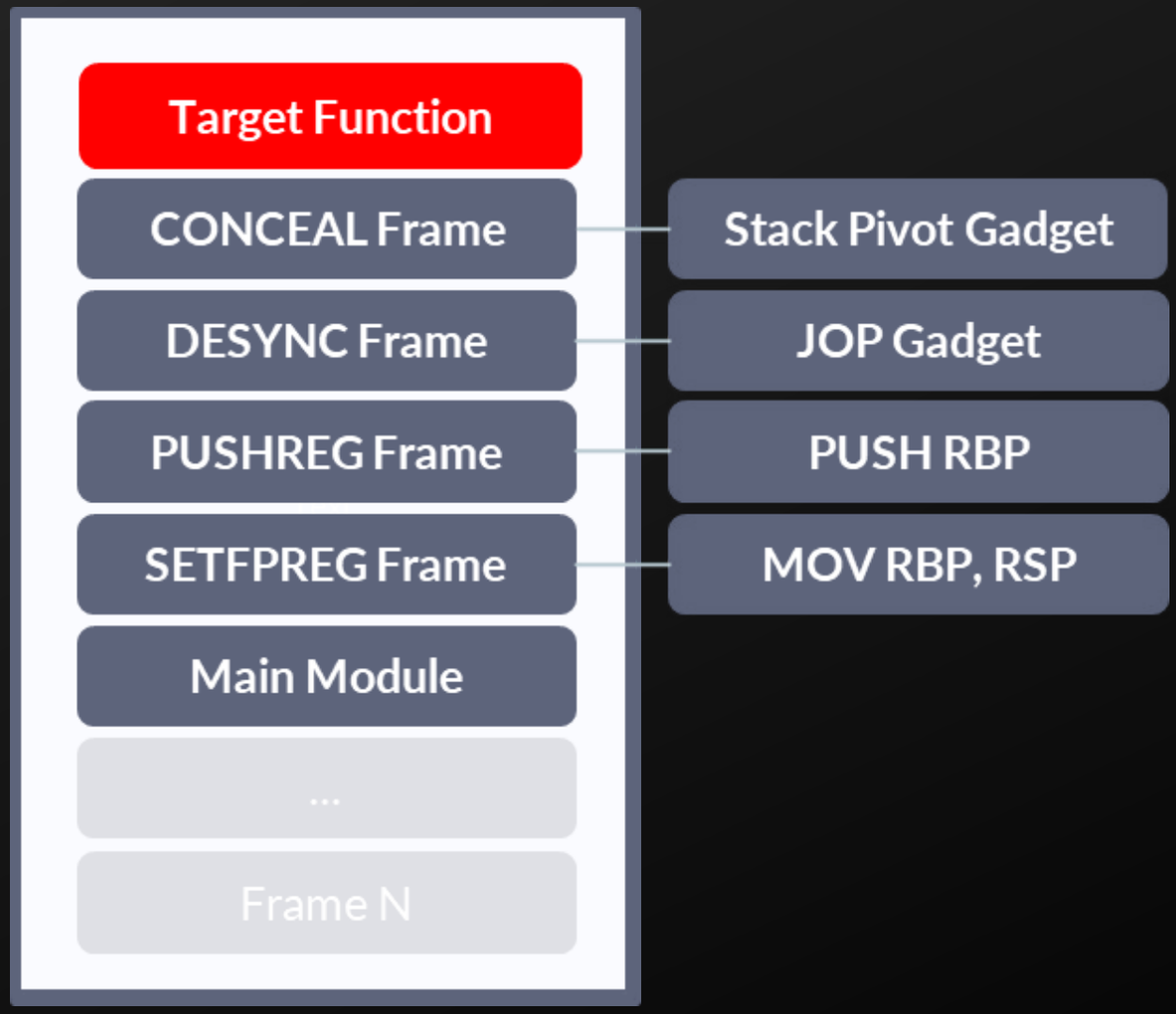
FULL MOON (WALK)



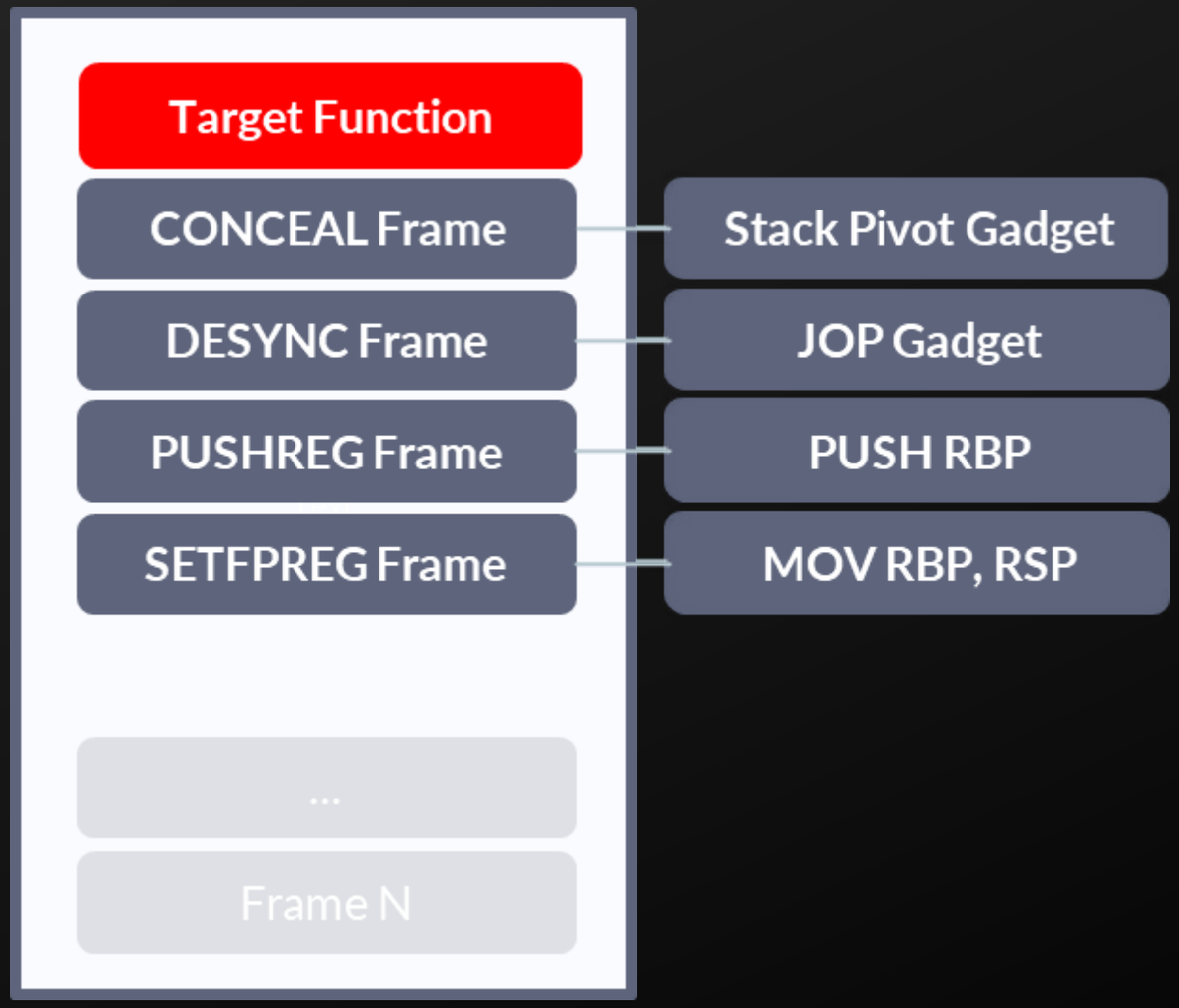
FULL MOON (WALK)



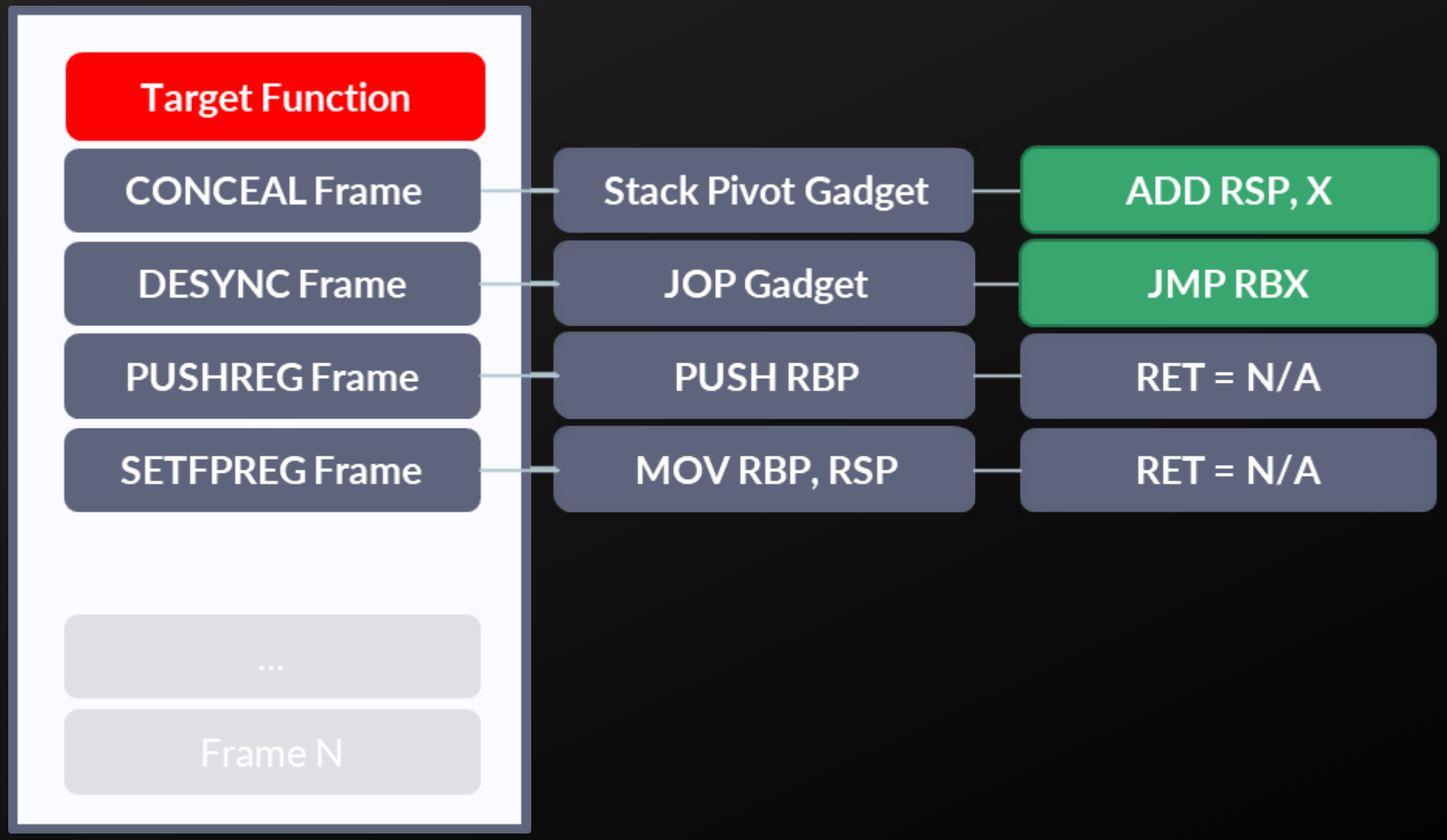
FULL MOON (WALK)



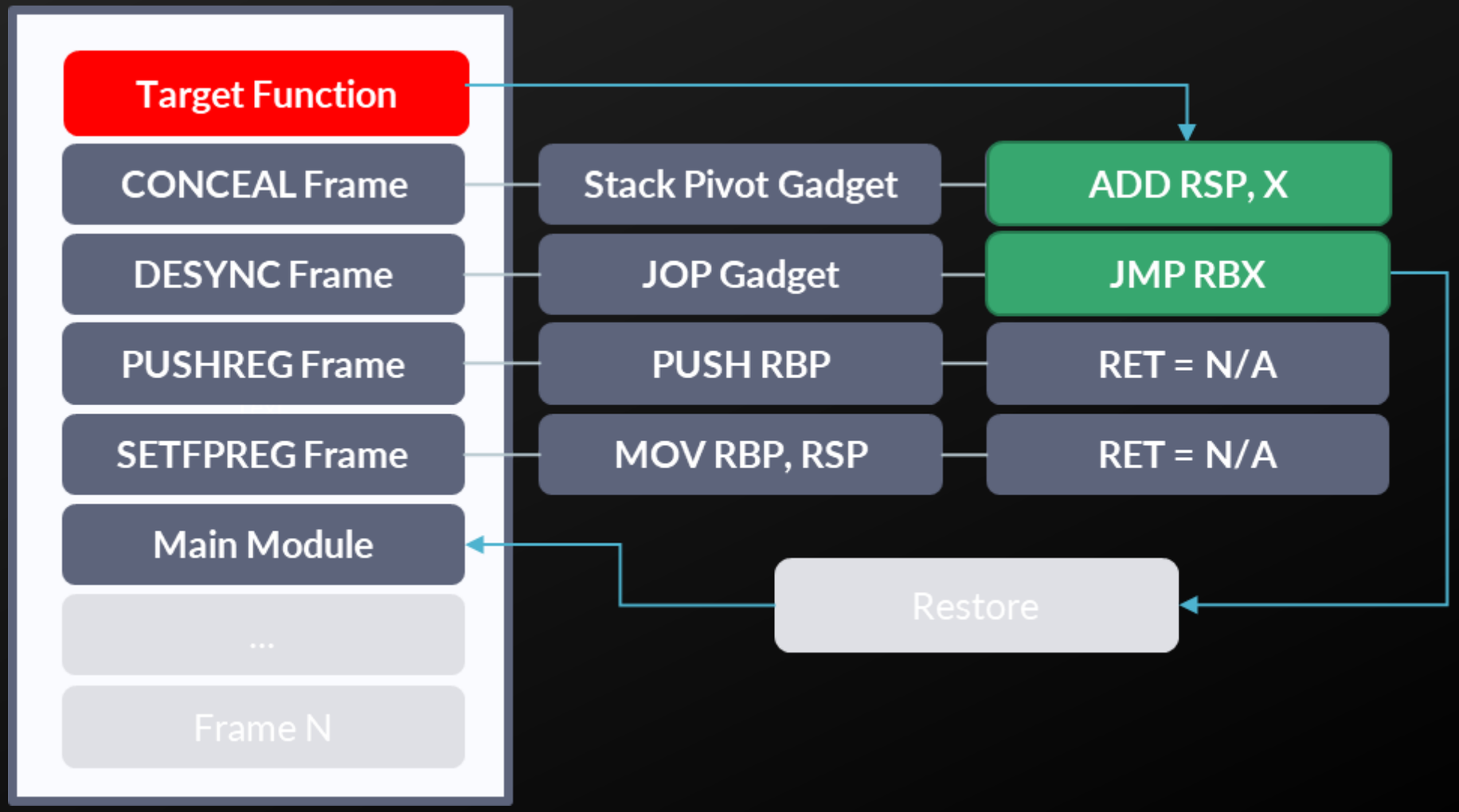
FULL MOON (WALK)



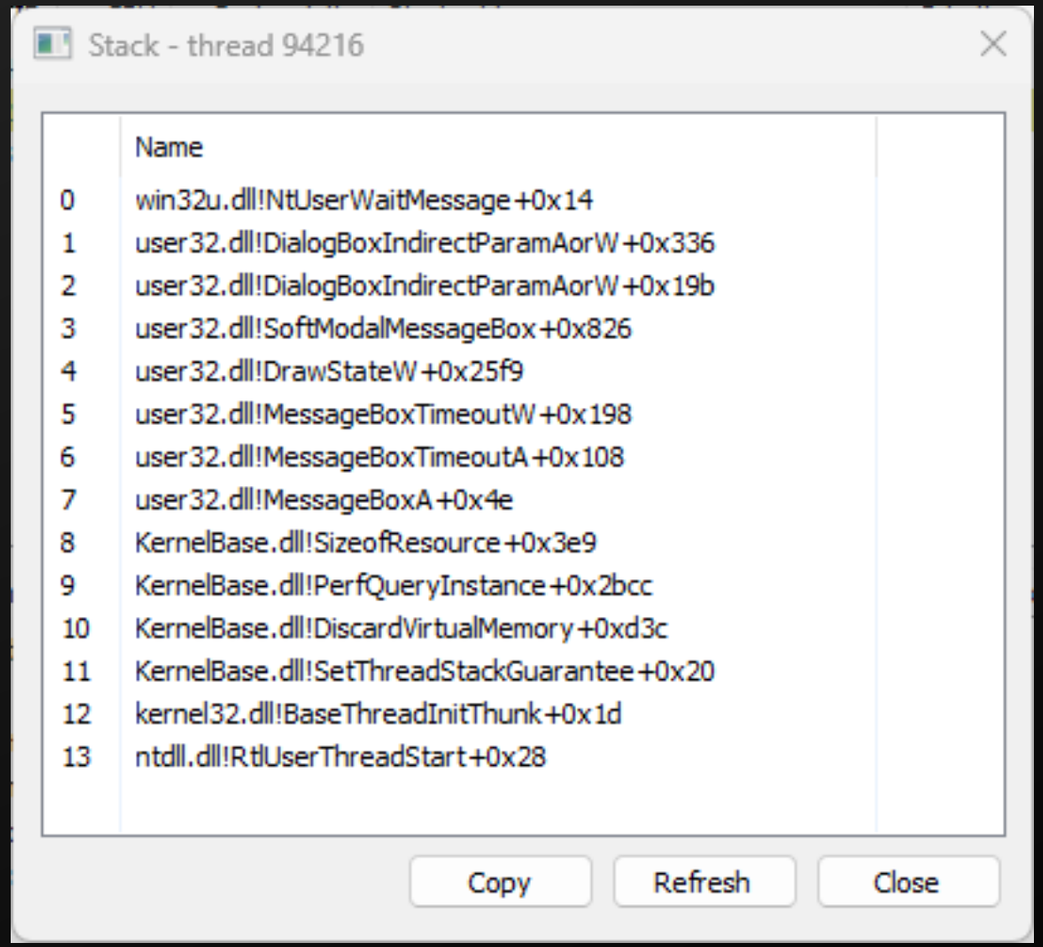
FULL MOON (WALK)



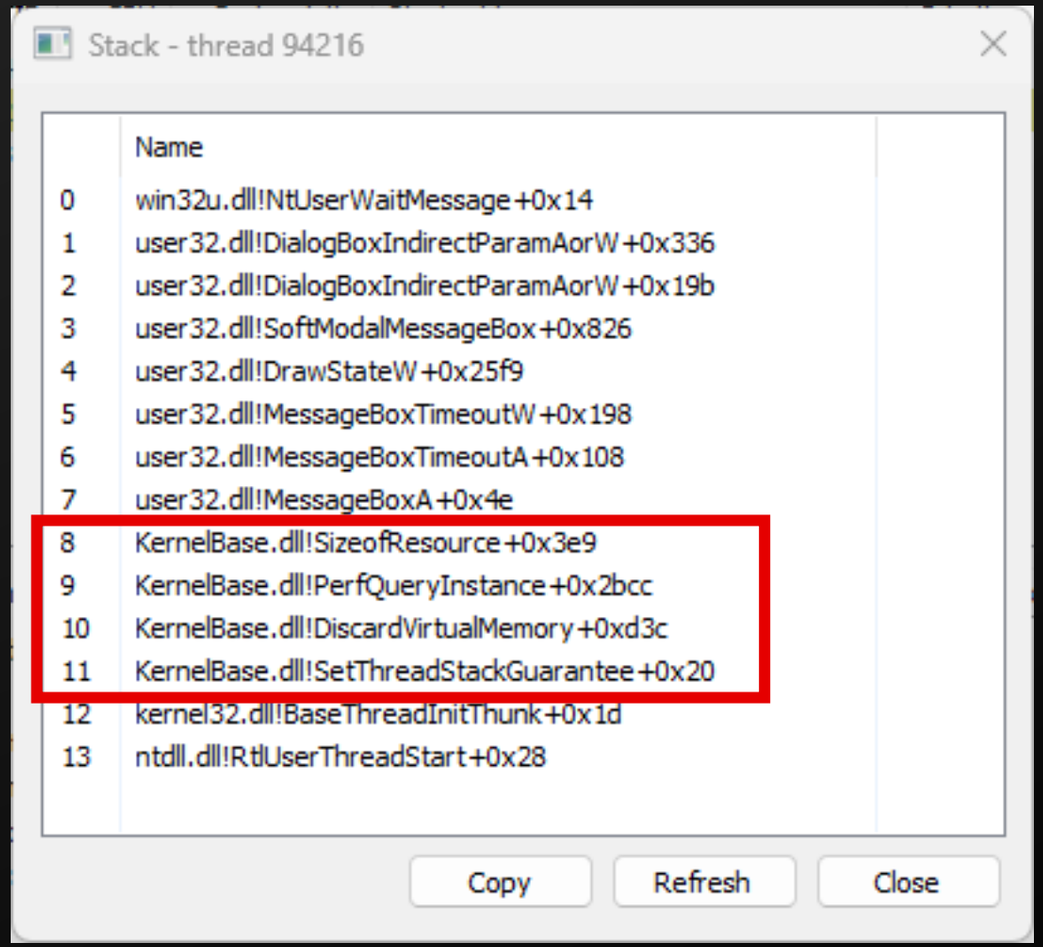
FULL MOON (WALK)



FULL MOON (WALK)



FULL MOON (WALK) - FRAMES



ECLIPSE


0:000> u user32!MessageBoxTimeoutA+0x108 - 5 L1
USER32!MessageBoxTimeoutA+0x103:
00007ffe`1961c723 e868000000 call USER32!MessageBoxTimeoutW (00007ffe`1961c790)
Legitimate Frame
0, win32u.dll!NtUserWaitMessage+0x14
1, user32.dll!FrameRect+0x27c
2, user32.dll!EndDialog+0x65b
3, user32.dll!SoftModalMessageBox+0x836
4, user32.dll!DrawStateA+0x1e11
5, user32.dll!MessageBoxTimeoutW+0x198
6, user32.dll!MessageBoxTimeoutA+0x108
7, user32.dll!MessageBoxA+0x4e
8, KernelBase.dll!CreatePrivateObjectSecurityWithMultipleInheritance+0x58
9, KernelBase.dll!OpenStateExplicit+0x39d
10, KernelBase.dll!DiscardVirtualMemory+0x39
11, KernelBase.dll!DiscardVirtualMemory+0x362
12, kernel32.dll!BaseThreadInitThunk+0x10
13, ntdll.dll!RtlUserThreadStart+0x2b
Malicious Frame
0:000> u KernelBase!OpenStateExplicit+0x39d - 5 L1
KERNELBASE!ReservedForLocalUse::HashThePublisher<unsigned short,unsigned short const *>+0x4c:
00007ffe`18b20198 884ada mov byte ptr [rdx-26h],cl

ECLIPSE

Expected CALL Instructions

Instruction	Opcode	Length
CALL MODR/M Rex.W 1	48 FF cd	0x7
CALL MODR/M Rex.W 0	FF cd	0x6
CALL REL32	E8 cd	0x5
CALL R64	[41] FF cb	0x2 - 0x3
SYSTEM CALL	0F 05	0x2

Possible CALL instructions observable before a frame return address

ECLIPSE - ALGORITHM

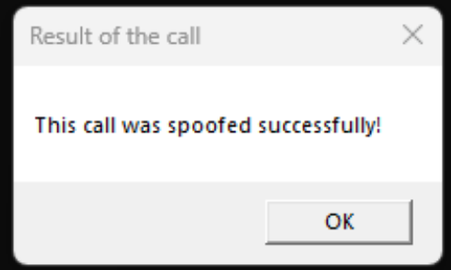
Algorithm 2: Eclipse detection algorithm

Result: True if $Eclipse(f) = 0 \; \forall f \; \in s$
Input : s: Stack, current: Frame, strict: Boolean
Output: alarm: Integer

```
1  while s is not empty do
2      frame ← s.pop()
3      prev_s ← prev_call_instruction(frame)
4      retn_s ← instruction_on_return(frame)
5      if frame is not backed by a file on disk then
6          return REFLECTIVE_INJECTION_DETECTED
7      else if ret_s == DESYNC GADGET then
8          return STACK_MOONWALK_DETECTED
9      else if prev_s! = CALL then
10         return CALL_MISMATCH_DETECTED
11     else
12         address ← extract(prev_s)
           if address does not call current ∧ strict then
13             return CALL_WRONG_ADDRESS_DETECTED
14         else
15             current ← frame
16         end
17     end
18     return NO_ALERT;
19
```

ECLIPSE - IN ACTION

```
Function BaseThreadInitThunk found. Stack size: 0x28 - Address: 0x7ffef88b2690
Function RtlUserThreadStart found. Stack size: 0x78 - Address: 0x7ffef988aa40
Runtime Function Table Size: 102312
Runtime Function Table Last Index: 8526
RT Function Table Range: 0x7FFEF7418000 - 0x7FFEF7430FA8
Return address: 0x244C8FFBD8
Address of Function to spoof: 0x7FFEF8176DB0

  ---------------------------------
Breaking at: 74
First Frame FP: 0x7FFEF70C54B4
First Frame stack size: 0x58
Return address: 0x7FFEF70C0160
Breaking at: 2
Second Frame FP: 0x7FFEF70C1470
Second Frame stack size: 0x148
Return address: 0x7FFEF70C0520
PUSH RBP offset: 0x140
Breaking at: 632
Gadget Address: 0x7FFEF70E4BCB
JMP [RBX] Frame Stack size: 0x168
Breaking at: 15
Gadget Address: 0x7FFEF70C245F
ADD RSP, X Frame Stack size: 0x38
```



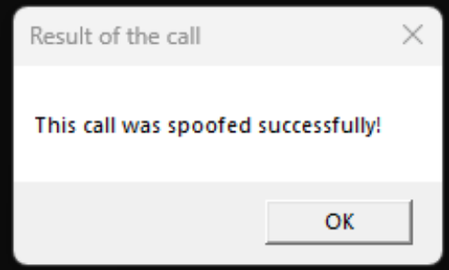
ECLIPSE - IN ACTION

```
Function BaseThreadInitThunk found. Stack size: 0x28 - Address: 0x7ffef88b2690
Function RtlUserThreadStart found. Stack size: 0x78 - Address: 0x7ffef988aa40
```

#	Name
0	win32u.dll!NtUserWaitMessage+0x14
1	user32.dll!DialogBox2+0x172
2	user32.dll!InternalDialogBox+0x127
3	user32.dll!SoftModalMessageBox+0x826
4	user32.dll!MessageBoxWorker+0x341
5	user32.dll!MessageBoxTimeoutW+0x198
6	user32.dll!MessageBoxTimeoutA+0x108
7	user32.dll!MessageBoxA+0x4e
8	KernelBase.dll!wistd::function<long __cdecl(unsigned short *
9	KernelBase.dll!CreateFileInternal+0x34b
10	KernelBase.dll!PssQuerySnapshot+0x25
11	KernelBase.dll!QueryProcessMachine+0x82
12	kernel32.dll!BaseThreadInitThunk+0x1d
13	ntdll.dll!RtlUserThreadStart+0x28

```
Gadget Address: 0x7FFEF70E4BCB
JMP [RBX] Frame Stack size: 0x168
Breaking at: 15
Gadget Address: 0x7FFEF70C245F
ADD RSP, X Frame Stack size: 0x38
```

```
Analyzing ThreadID: 4436

Analyzed Function Frame: CreateFileW
 PC Address: 0x00007FFEF70E4BCB
 Stack Address: 0x000000244C8FF820
 Return Address: 0x00007FFEF70C1495
[X] Possibly spoofed return address observed!  There is a false caller observed here.
[!] This alert was generated because a call was not observed before the return address
 UNLESS this is a JIT process (such as a C Sharp Process).  Then FP's are expected.
[Information] Opcode observed at return address is: 0xF1
```



ECLIPSE - IN ACTION

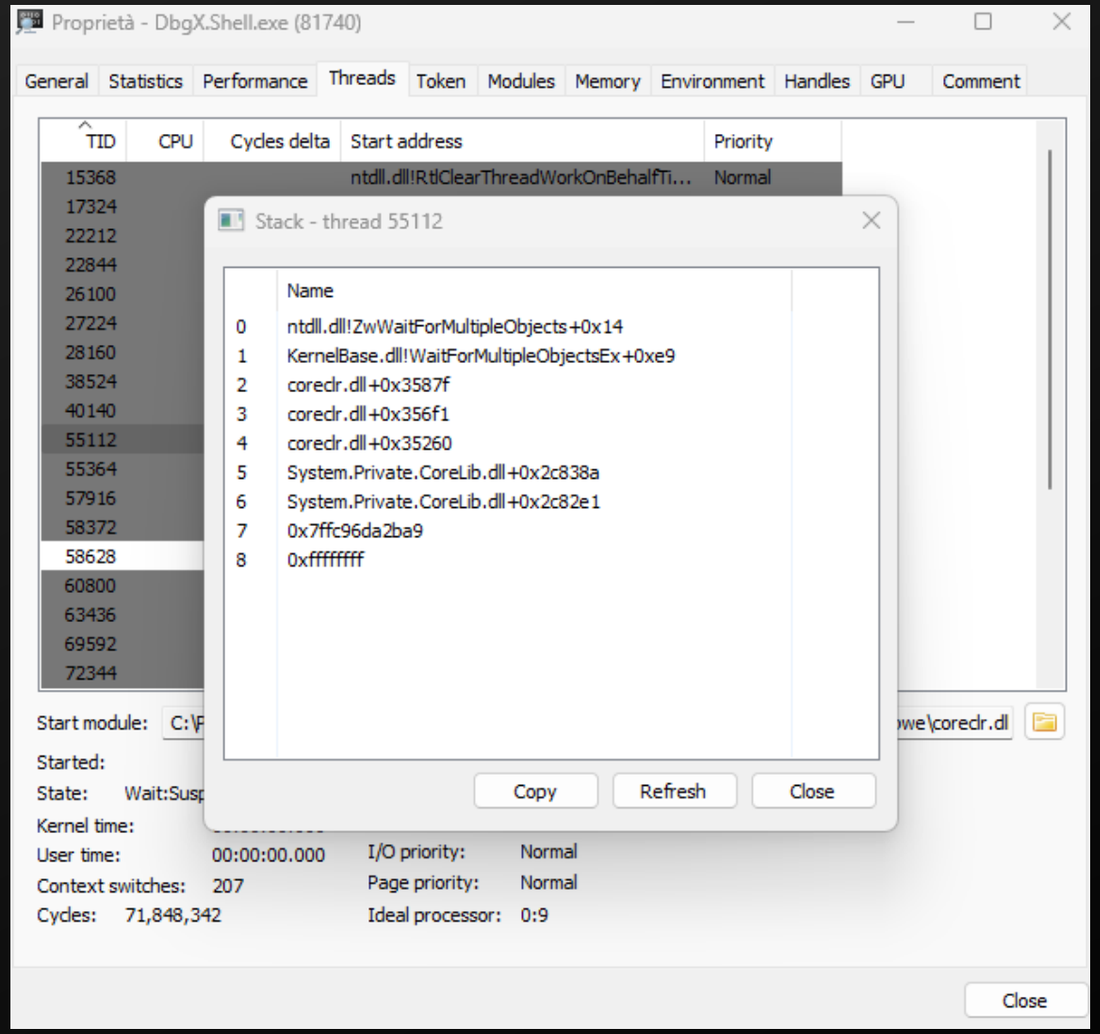
```
Function BaseThreadInitThunk found. Stack size: 0x28 - Address: 0x7ffef88b2690
Function RtlUserThreadStart found. Stack size: 0x78 - Address: 0x7ffef988aa40
```

#	Name
0	win32u.dll!NtUserWaitMessage+0x14
1	user32.dll!DialogBox2+0x172
2	user32.dll!InternalDialogBox+0x127
3	user32.dll!SoftModalMessageBox+0x826
4	user32.dll!MessageBoxWorker+0x341
5	user32.dll!MessageBoxTimeoutW+0x198
6	user32.dll!MessageBoxTimeoutA+0x108
7	user32.dll!MessageBoxA+0x4e
8	KernelBase.dll!wistd::function<long __cdecl(unsigned short *
9	KernelBase.dll!CreateFileInternal+0x34b
10	KernelBase.dll!PssQuerySnapshot+0x25
11	KernelBase.dll!QueryProcessMachine+0x82
12	kernel32.dll!BaseThreadInitThunk+0x1d
13	ntdll.dll!RtlUserThreadStart+0x28

```
Analyzing ThreadID: 4436

Analyzed Function Frame: CreateFileW
 PC Address: 0x00007FFEF70E4BCB
 Stack Address: 0x000000244C8FF820
 Return Address: 0x00007FFEF70C1495
[X] Possibly spoofed return address observed!  There is a false caller observed here.
[!] This alert was generated because a call was not observed before the return address
 UNLESS this is a JIT process (such as a C Sharp Process).  Then FP's are expected.
[Information] Opcode observed at return address is: 0xF1

Analyzed Function Frame: PssQuerySnapshot
 PC Address: 0x00007FFEF70C1495
 Stack Address: 0x000000244C8FF990
 Return Address: 0x00007FFEF70C5536
[X] Possibly spoofed return address observed!  There is a false caller observed here.
[!] This alert was generated because a call was not observed before the return address
 UNLESS this is a JIT process (such as a C Sharp Process).  Then FP's are expected.
[Information] Opcode observed at return address is: 0x0F
```

```
Gadget Address: 0x7FFEF70E4BCB
JMP [RBX] Frame Stack size: 0x168
Breaking at: 15
Gadget Address: 0x7FFEF70C245F
ADD RSP, X Frame Stack size: 0x38
```

ECLIPSE - FALSE POSITIVES



ECLIPSE - ACCURACY AND JIT

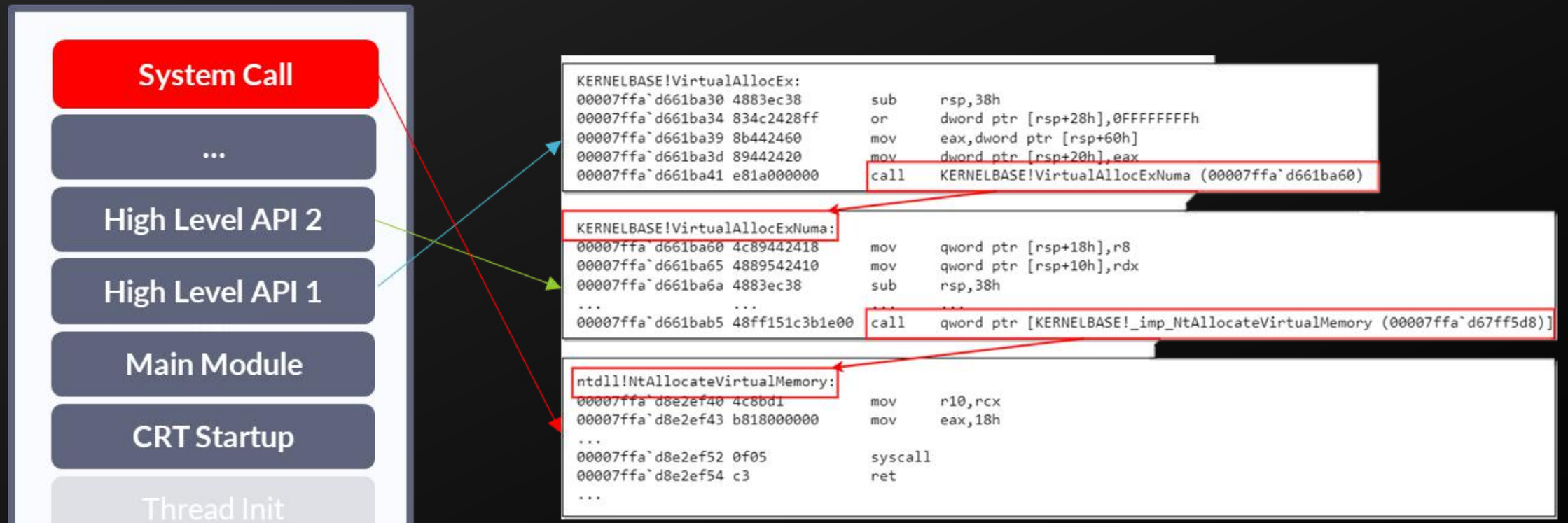
Machine	Processes Count	Alerts	False Positive Rate
Windows 10 Pro (std)	190	2	1.05%
Windows 10 Pro (dev)	350	15	4.29%
Windows 11 Enterprise (std)	176	4	2.27%
Windows 11 Enterprise (dev)	330	12	3.64%
Windows Server 2019	160	5	3.13%

Number of Processes

Number of processes with anomalies

False Positives

HALF MOON (WALK)



HALF MOON (WALK)

```
0, ntdll.dll!NtAllocateVirtualMemory+0x13
1, KernelBase.dll!VirtualAllocExNuma+0x5c
2, KernelBase.dll!VirtualAllocEx+0x16
3, AbsentMoon.exe!main+0x5a
4, AbsentMoon.exe!__scrt_common_main_seh+0x10c
5, kernel32.dll!BaseThreadInitThunk+0x1d
6, ntdll.dll!RtlUserThreadStart+0x28
```

Normal

```
0, ntdll.dll!NtAllocateVirtualMemory+0x13
1, KernelBase.dll!VirtualAllocExNuma+0x5c
2, KernelBase.dll!VirtualAllocEx+0x16
3, HalfMoon.exe!emulate_system_call+0x3f
4, HalfMoon.exe!main+0x322
5, HalfMoon.exe!__scrt_common_main_seh+0x10c
6, kernel32.dll!BaseThreadInitThunk+0x1d
7, ntdll.dll!RtlUserThreadStart+0x28
```

Half Moon

HALF MOON (WALK) - RETURN FLOW

1

Place Return on Unwinding

It is possible to set the return address at the start of the epilogue, which can be calculated using the RUNTIME_FUNCTION information.

2

Targeted Setup

It is also possible to analyze and tamper the post-return checks to ensure the program returns without errors.

3

Sliding HW breakpoints

It is possible to use a "sliding breakpoint" strategy to force the return flow in a suitable way.

HALF MOON (WALK) - RETURN ON EPILOG

```
KERNELBASE!VirtualProtectEx:
00007fff`f798de10 488bc4          mov     rax,rsp
00007fff`f798de13 48895808        mov     qword ptr [rax+8],rbx
00007fff`f798de17 4c894018        mov     qword ptr [rax+18h],r8
00007fff`f798de1b 48895010        mov     qword ptr [rax+10h],rdx
00007fff`f798de1f 55              push    rbp
00007fff`f798de20 56              push    rsi
00007fff`f798de21 57              push    rdi
00007fff`f798de22 4883ec30        sub     rsp,30h
00007fff`f798de26 488b6c2470      mov     rbp,qword ptr [rsp+70h]
00007fff`f798de2b 4c8d4018        lea     r8,[rax+18h]
00007fff`f798de2f 488d5010        lea     rdx,[rax+10h]
00007fff`f798de33 488968d8        mov     qword ptr [rax-28h],rbp
00007fff`f798de37 418bf1          mov     esi,r9d
00007fff`f798de3a 488bf9          mov     rdi,rcx
00007fff`f798de3d 48ff15b4171e00  call    qword ptr [KERNELBASE!_imp_NtProtectVirtualMemory (00007fff`f7b6f5f8)]
00007fff`f798de44 0f1f440000      nop     dword ptr [rax+rax]
00007fff`f798de49 8bd8            mov     ebx,eax
00007fff`f798de4b 85c0            test    eax,eax
00007fff`f798de4d 0f88ab510700    js      KERNELBASE!VirtualProtectEx+0x751ee (00007fff`f7a02ffe)
00007fff`f798de53 b801000000      mov     eax,1
00007fff`f798de58 488b5c2450      mov     rbx,qword ptr [rsp+50h]
00007fff`f798de5d 4883c430        add     rsp,30h
00007fff`f798de61 5f              pop     rdi
00007fff`f798de62 5e              pop     rsi
00007fff`f798de63 5d              pop     rbp
00007fff`f798de64 c3              ret
```

HALF MOON (WALK) - TARGETED SETUP

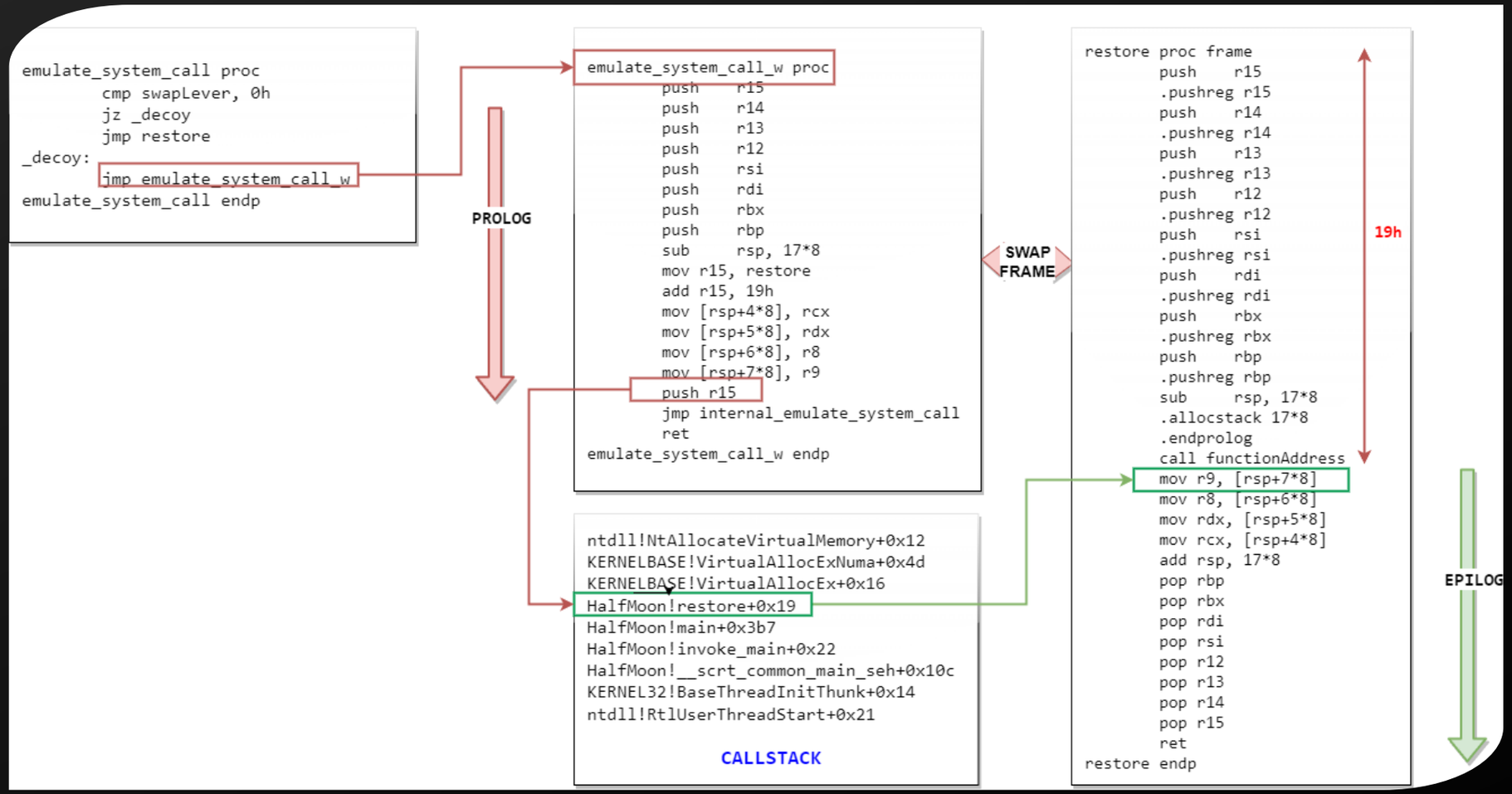
```
execute proc
    push        lastFrameCall
    xor         rsi, rsi
    mov         rbp, offset dummy
    mov         [rbp+07fh], rsi
    xor         r14, r14
    xor         r12, r12
    push        rbx
    xor         rbx, rbx
    ret
execute endp
```

```
KERNELBASE!WriteProcessMemory+0xb8:
00007fff`f7989ed8 48ff1521571e00  call    qword ptr [KERNELBASE!_imp_NtWriteVirtualMemory (00007fff`f7b6f600)]
00007fff`f7989edf 0f1f440000      nop     dword ptr [rax+rax]
00007fff`f7989ee4 488b4d7f        mov     rcx,qword ptr [rbp+7Fh]
00007fff`f7989ee8 8bd8            mov     ebx,eax
00007fff`f7989eea 4885c9          test    rcx,rcx
00007fff`f7989eed 754e            jne     KERNELBASE!WriteProcessMemory+0x11d (00007fff`f7989f3d)
...
00007fff`f7989f29 4881c4a0000000  add     rsp,0A0h
00007fff`f7989f30 415f            pop     r15
00007fff`f7989f32 415e            pop     r14
00007fff`f7989f34 415d            pop     r13
00007fff`f7989f36 415c            pop     r12
00007fff`f7989f38 5f              pop     rdi
00007fff`f7989f39 5e              pop     rsi
00007fff`f7989f3a 5d              pop     rbp
00007fff`f7989f3b c3              ret
```

HALF MOON (WALK) - SLIDING HWBP



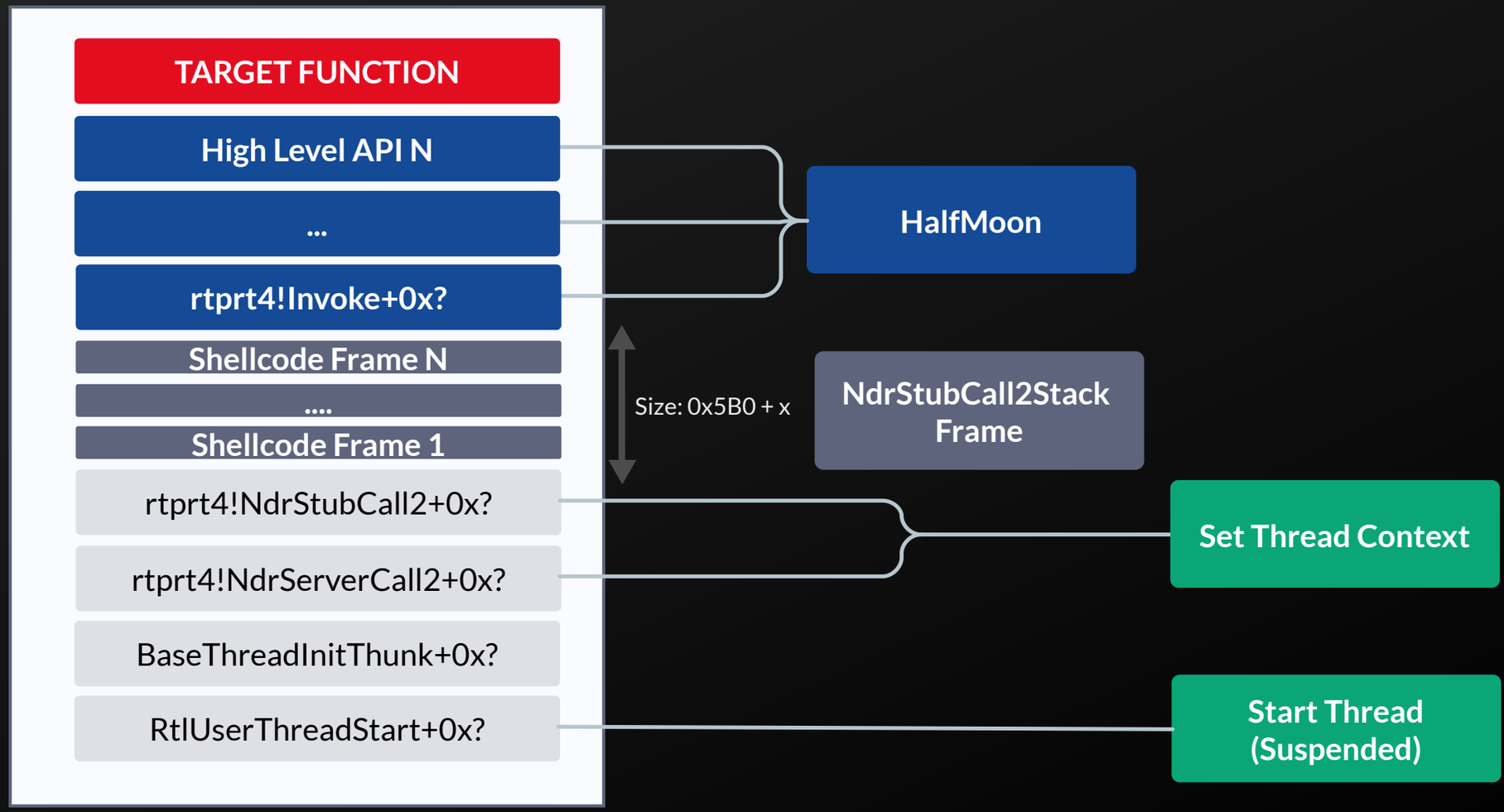
OPAQUE ARCHITECTURE



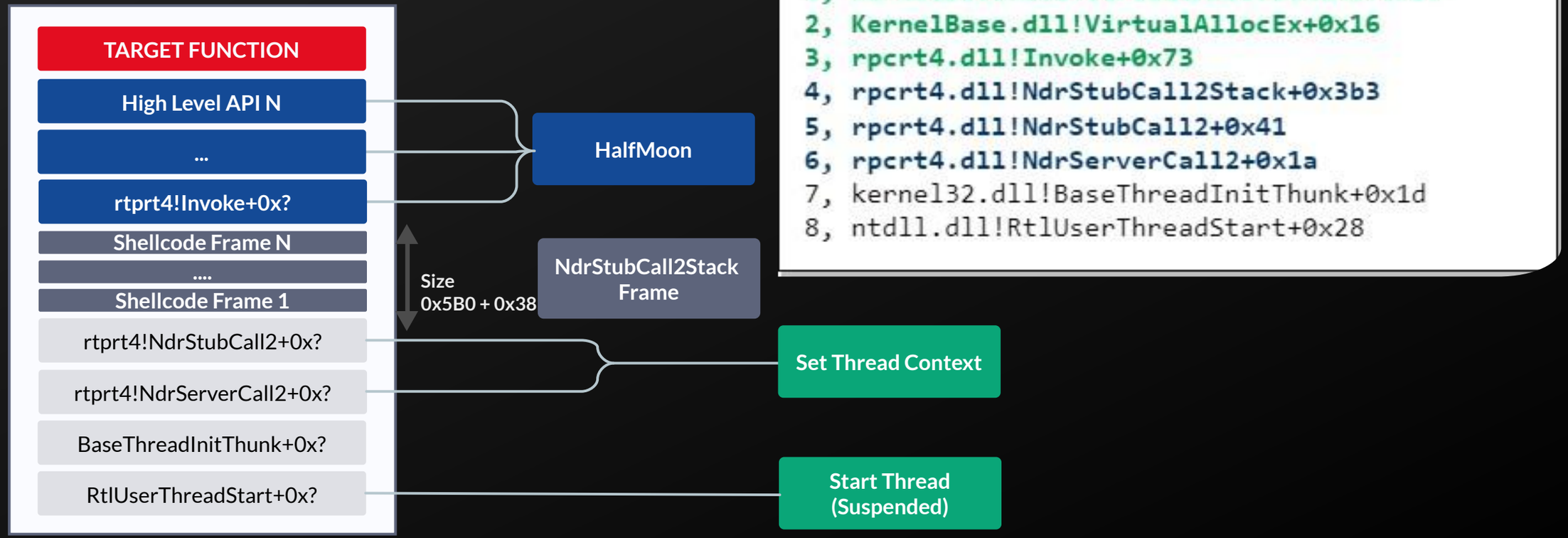
OPAQUE ARCHITECTURE - RPC

```
RPCRT4!NdrServerCall2:
00007fff`f8a50380 4883ec28        sub     rsp,28h
...
00007fff`f8a50395 e816be0300      call    RPCRT4!NdrStubCall2 (00007fff`f8a8c1b0)

RPCRT4!NdrStubCall2:
...
00007fff`f8a8c1bf 57              push    rdi
00007fff`f8a8c1c0 4883ec20        sub     rsp,20h
...
00007fff`f8a8c1ec e857a50100      call    RPCRT4!NdrStubCall2Stack (00007fff`f8aa6748)

RPCRT4!NdrStubCall2Stack:
...
00007fff`f8aa6754 4881ecb0050000  sub     rsp,5B0h
...
00007fff`f8aa6af6 e885080000      call    RPCRT4!Invoke (00007fff`f8aa7380)

RPCRT4!Invoke:
00007fff`f8aa7380 4883ec38        sub     rsp,38h
...
00007fff`f8aa7393 488bec          mov     rbp,rsp
...
00007fff`f8aa73f0 41ffd2          call    r10
```

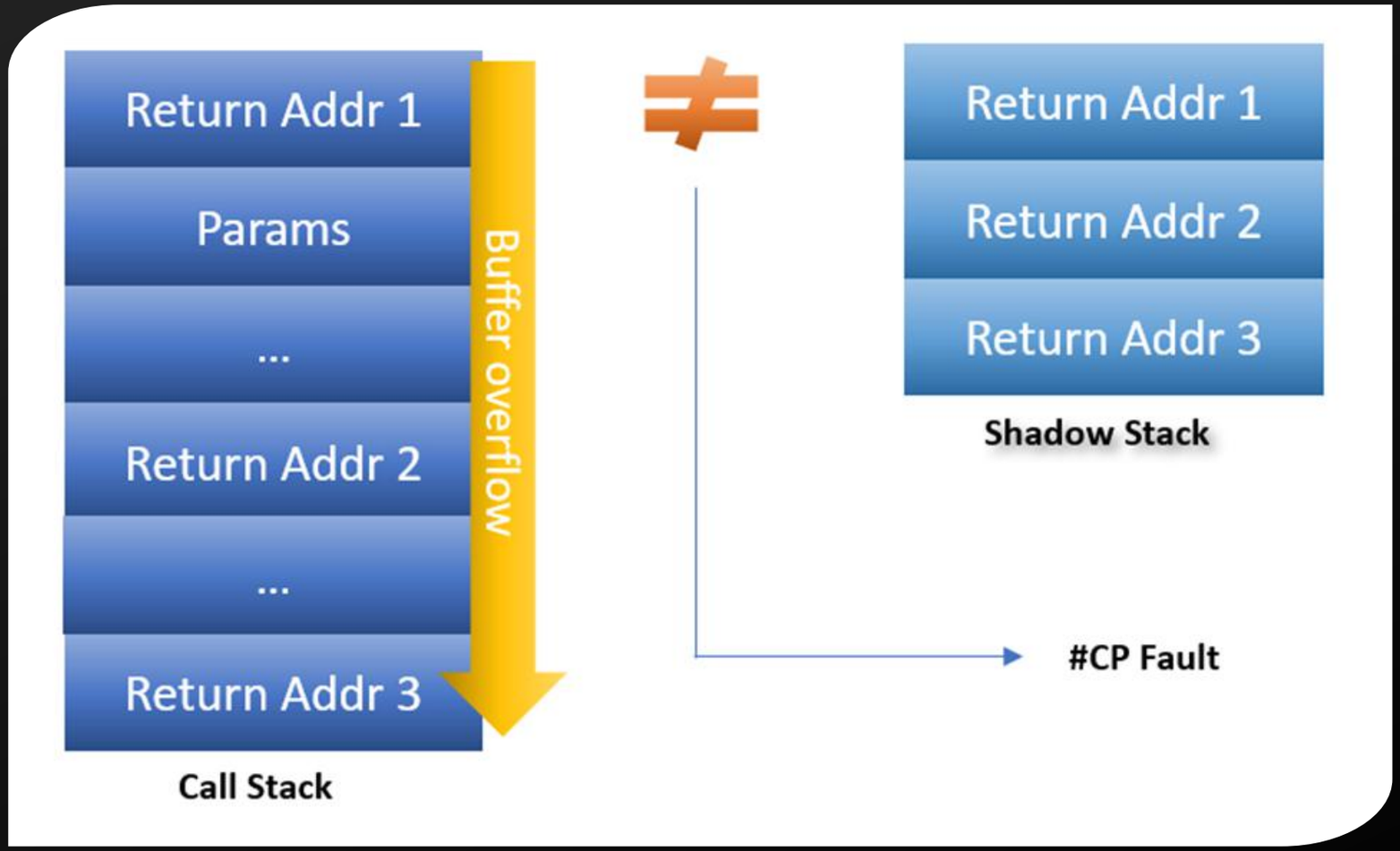
OPAQUE ARCHITECTURE - RPC



OPAQUE ARCHITECTURE - RPC



ABOUT INTEL CET



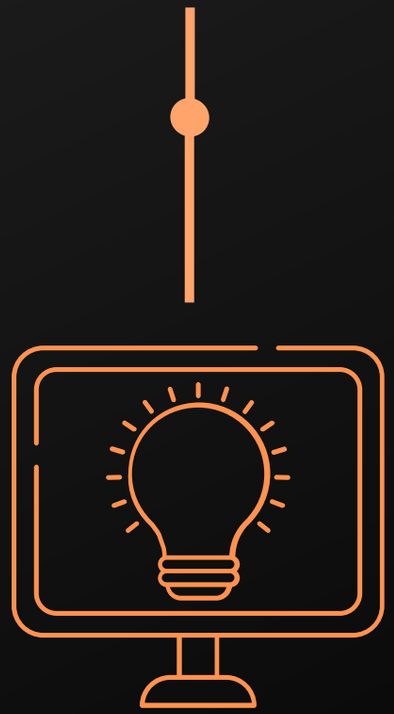
KEY TAKEAWAYS

Public POC Available
https://github.com/klezVirus/SilentMoonwalk

It adds a layer of complexity which is not always easy to manage

It is possible to create chains of different length by inserting stack pivot frames

This POC is designed to obfuscate the stack at runtime, and can be used wherever return address spoofing is used



Full Moon obfuscate the caller frames but creates inconsistences across frames. Half Moon and its variations, instead, require executable memory (Exception Handlers) to restore the stack on return

For sleep obfuscation an approach like Stack Hiding or Stack Cloning is preferred

It is possible to integrate full encryption of the code at runtime (i.e. sleep encryption) whenever an API is invoked


thanks!