confidence

JOES

Watch Out! And just skip the packer

Felipe Duarte
Senior Threat Researcher @ Security Joes
@dark0pcodes

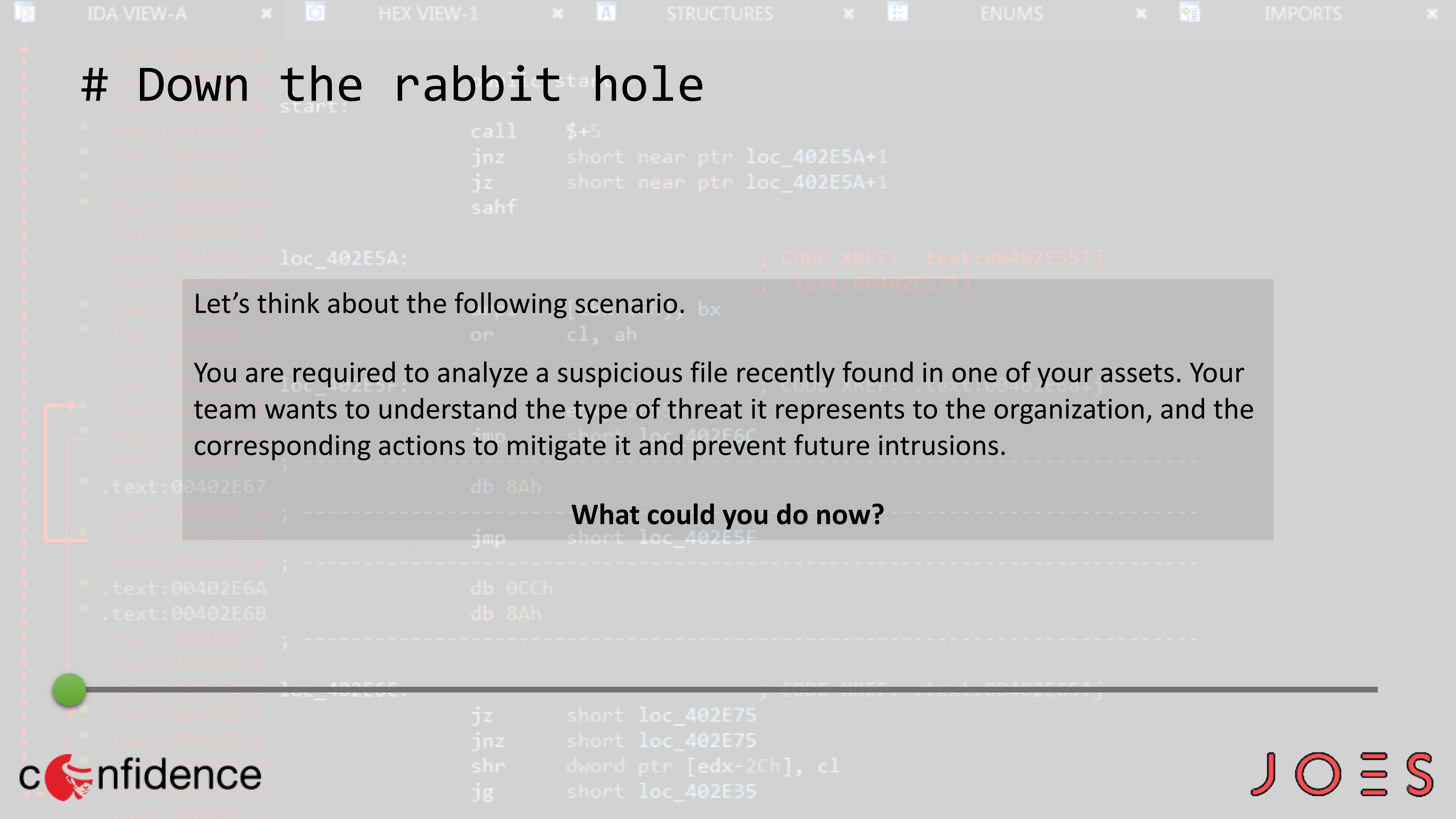
June 2022 - Krakow

Down the rabbit hole

Let’s think about the following scenario.

You are required to analyze a suspicious file recently found in one of your assets. Your team wants to understand the type of threat it represents to the organization, and the corresponding actions to mitigate it and prevent future intrusions.

What could you do now?



```
start:
                call    $+5
                jnz     short near ptr loc_402E5A+1
                jz      short near ptr loc_402E5A+1
                sahf

loc_402E5A:                             ; CODE XREF: .text:00402E55↑j
                                        ; .text:00402E57↑j
                or      cl, ah

loc_402E5F:                             ; CODE XREF: .text:00402E68↓j
                jmp     short loc_402E6C
.text:00402E67  db 8Ah
                jmp     short loc_402E5F
.text:00402E6A  db 0CCh
.text:00402E6B  db 8Ah

loc_402E6C:                             ; CODE XREF: .text:00402E65↑j
                jz      short loc_402E75
                jnz     short loc_402E75
                shr     dword ptr [edx-2Ch], cl
                jg      short loc_402E35
```

# Down the rabbit hole

1. Use a sandbox (public or private) and gather information via OSINT, but…

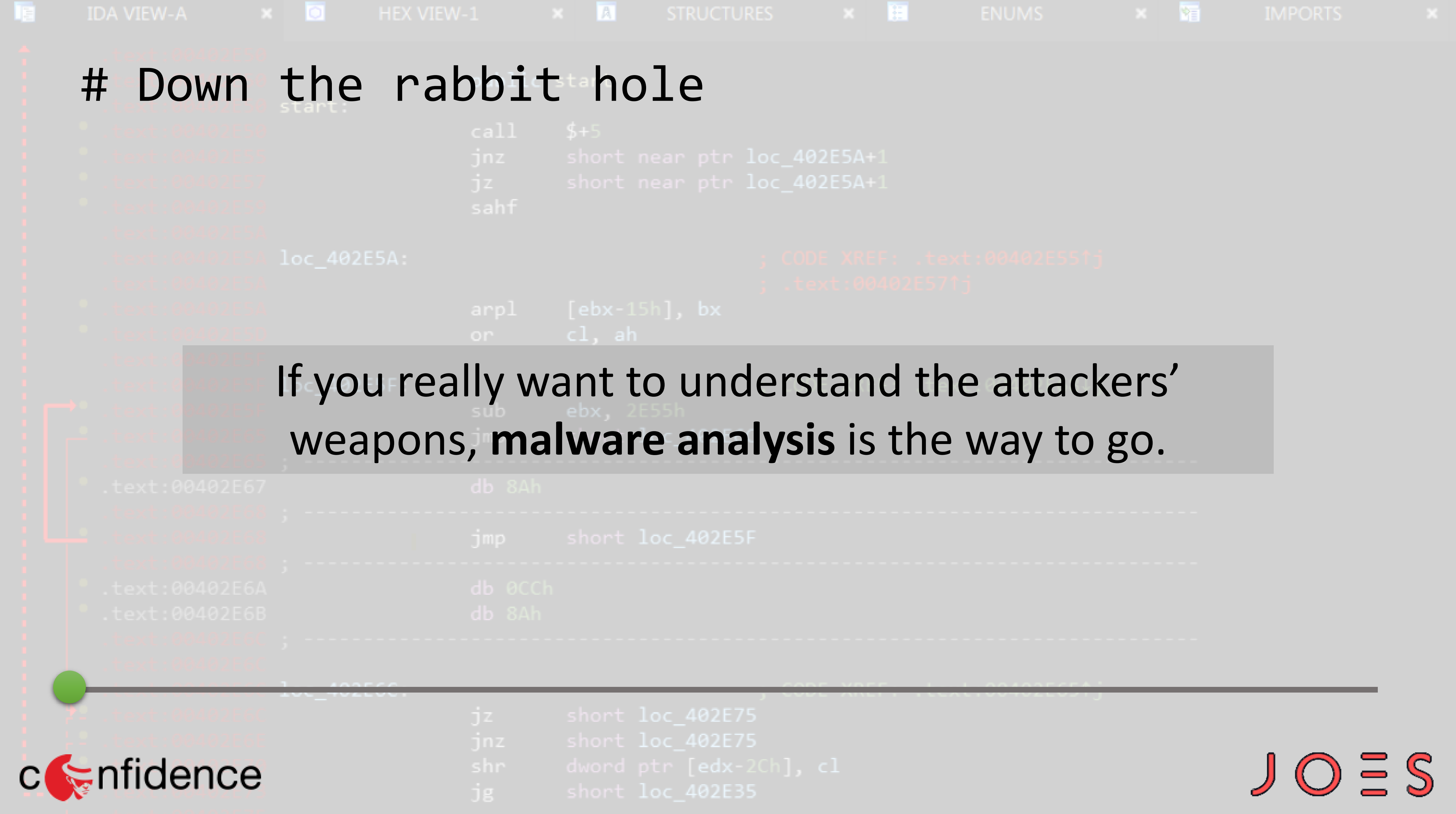
2. Do some memory forensics, but…

3. Dig deeper into the logs, but…

4. Do some manual malware analysis!

Down the rabbit hole

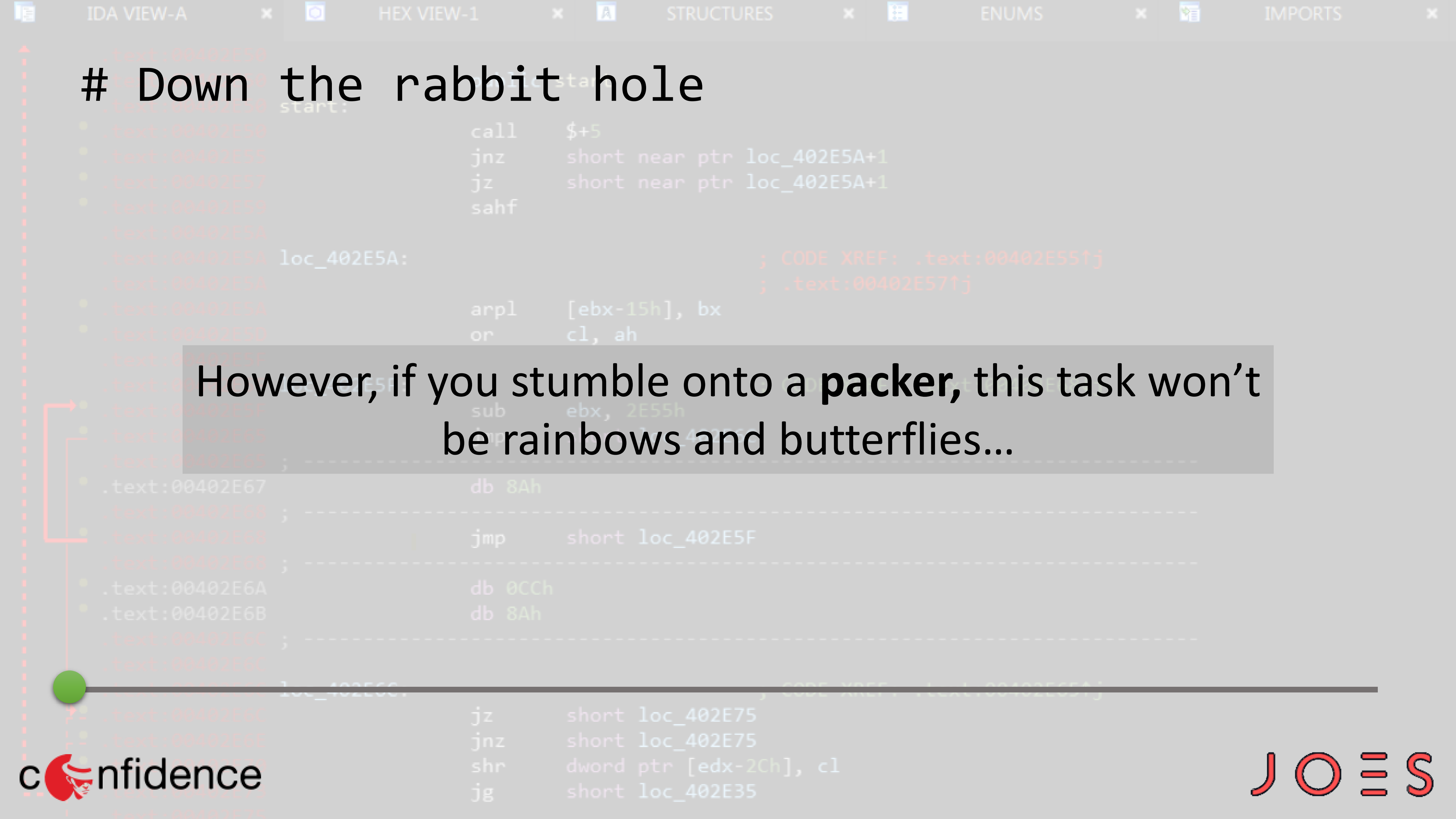
If you really want to understand the attackers' weapons, **malware analysis** is the way to go.



```
start:
                call    $+5
                jnz     short near ptr loc_402E5A+1
                jz      short near ptr loc_402E5A+1
                sahf

loc_402E5A:                             ; CODE XREF: .text:00402E55↑j
                                        ; .text:00402E57↑j
                arpl    [ebx-15h], bx
                or      cl, ah

                sub     ebx, 2E55h

.text:00402E67  db 8Ah

                jmp     short loc_402E5F

.text:00402E6A  db 0CCh
.text:00402E6B  db 8Ah

                jz      short loc_402E75
                jnz     short loc_402E75
                shr     dword ptr [edx-2Ch], cl
                jg      short loc_402E35
```

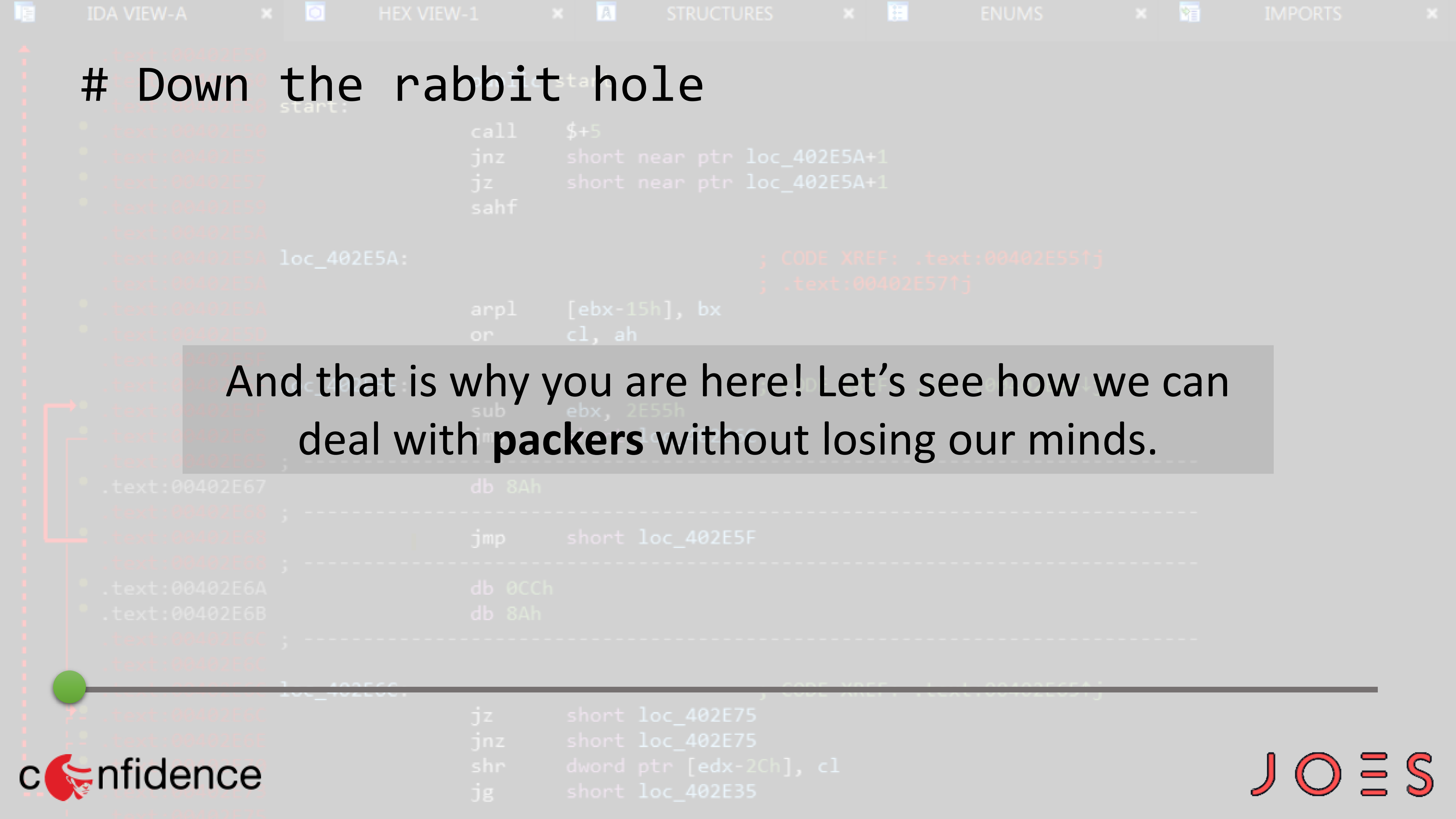
# Down the rabbit hole

However, if you stumble onto a **packer,** this task won’t be rainbows and butterflies…



```
start:
                call    $+5
                jnz     short near ptr loc_402E5A+1
                jz      short near ptr loc_402E5A+1
                sahf

loc_402E5A:                             ; CODE XREF: .text:00402E55↑j
                                        ; .text:00402E57↑j
                arpl    [ebx-15h], bx
                or      cl, ah

                sub     ebx, 2E55h

.text:00402E67  db 8Ah

                jmp     short loc_402E5F

.text:00402E6A  db 0CCh
.text:00402E6B  db 8Ah

                jz      short loc_402E75
                jnz     short loc_402E75
                shr     dword ptr [edx-2Ch], cl
                jg      short loc_402E35
```

# Down the rabbit hole



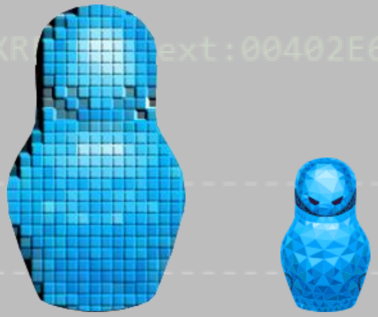
```
                     start:
                                    call    $+5
                                    jnz     short near ptr loc_402E5A+1
                                    jz      short near ptr loc_402E5A+1
                                    sahf

                     loc_402E5A:                                         ; CODE XREF: .text:00402E55↑j
                                                                         ; .text:00402E57↑j
                                    arpl    [ebx-15h], bx
                                    or      cl, ah

                                    sub     ebx, 2E55h

                     ; ---------------------------------------------------------------------------
.text:00402E67                      db 8Ah
                     ; ---------------------------------------------------------------------------
                                    jmp     short loc_402E5F
                     ; ---------------------------------------------------------------------------
.text:00402E6A                      db 0CCh
.text:00402E6B                      db 8Ah
                     ; ---------------------------------------------------------------------------

                     loc_402E6C:

                                    jz      short loc_402E75
                                    jnz     short loc_402E75
                                    shr     dword ptr [edx-2Ch], cl
                                    jg      short loc_402E35
```

And that is why you are here! Let’s see how we can deal with **packers** without losing our minds.

# What is a packer?

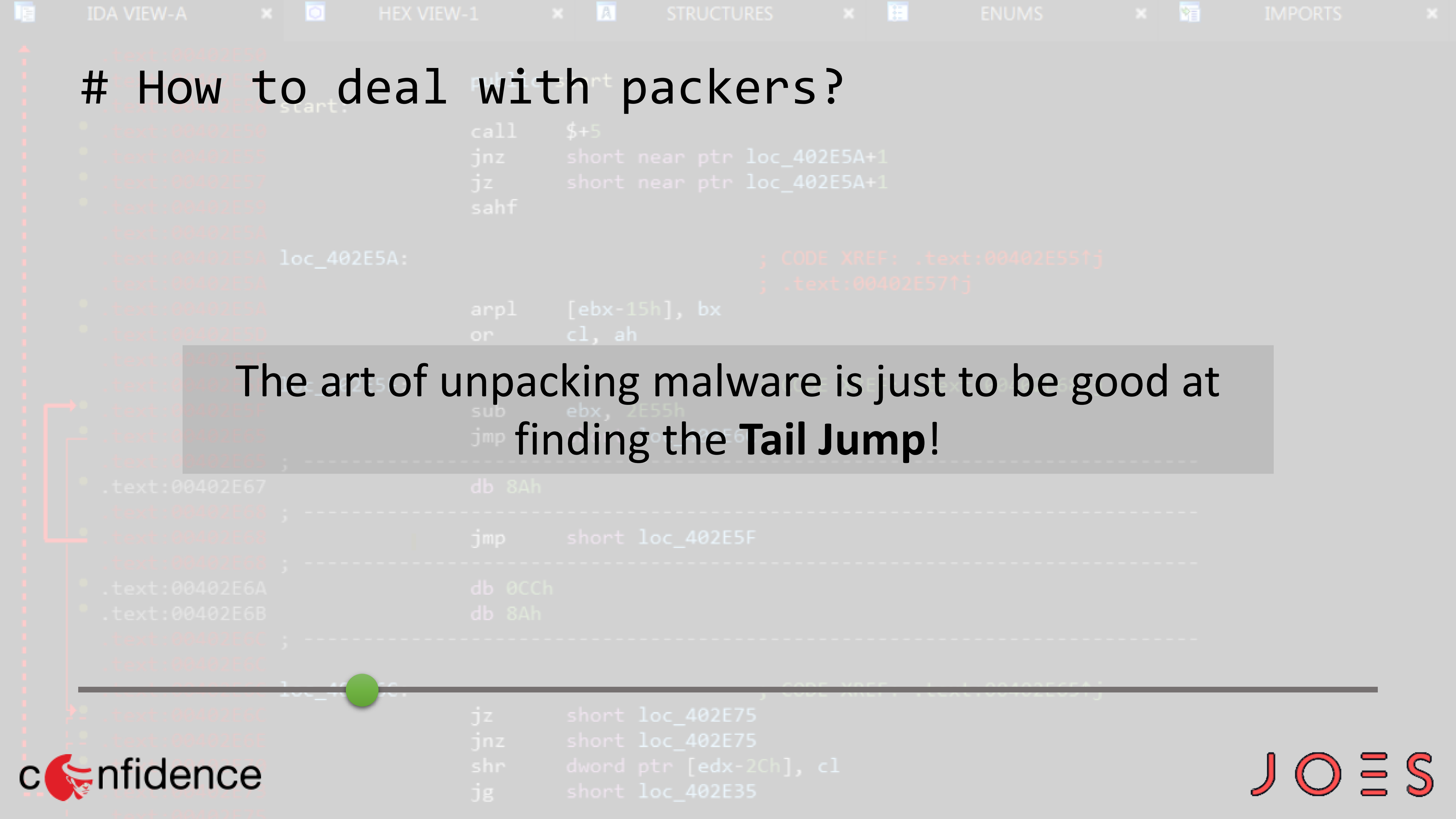
Tool used by software developers (malicious or not) to shield programs against reverse engineering. They provide different protections including but not limited to:

1. Anti-analysis validations
2. Polymorphism to avoid detection
3. Code obfuscation/encryption



# How to deal with packers?

The art of unpacking malware is just to be good at finding the **Tail Jump**!



```
                 public start
                 start:
                                call    $+5
                                jnz     short near ptr loc_402E5A+1
                                jz      short near ptr loc_402E5A+1
                                sahf

                 loc_402E5A:                            ; CODE XREF: .text:00402E55↑j
                                                        ; .text:00402E57↑j
                                arpl    [ebx-15h], bx
                                or      cl, ah

                                sub     ebx, 2E55h
                                jmp
                 ; ---------------------------------------------------------------
.text:00402E67                  db 8Ah
                 ; ---------------------------------------------------------------
                                jmp     short loc_402E5F
                 ; ---------------------------------------------------------------
.text:00402E6A                  db 0CCh
.text:00402E6B                  db 8Ah
                 ; ---------------------------------------------------------------

                                jz      short loc_402E75
                                jnz     short loc_402E75
                                shr     dword ptr [edx-2Ch], cl
                                jg      short loc_402E35
```

# What about the **Tail Jump?**

Instruction in which the execution of the packer ends, and the control flow is redirected to the entry point of the original unpacked sample. This jump can be implemented in several different ways, including but not limited to:

1 `JMP OEP_ADDRESS`

2 `CALL OEP_ADDRESS`

3 `PUSH OEP_ADDRESS -> RET`

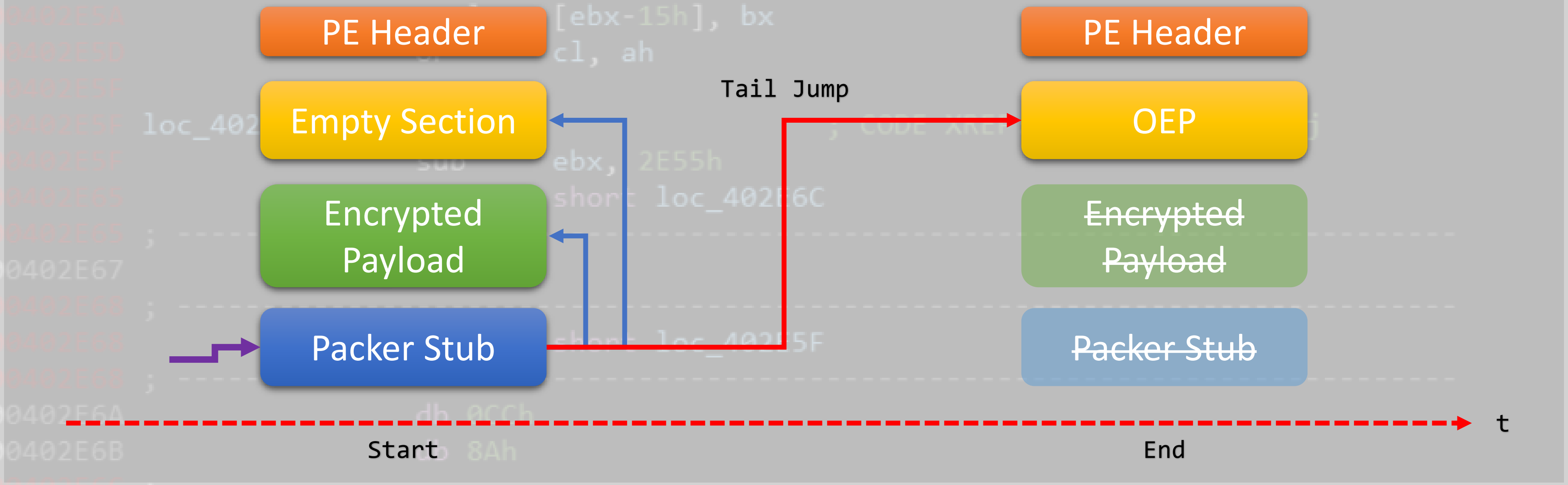
Types of packers

According to their behavior, packers can be classified into the following categories:

1. Code substitution packers
2. Code injection packers
3. Hybrid packers
4. Code virtualization packers

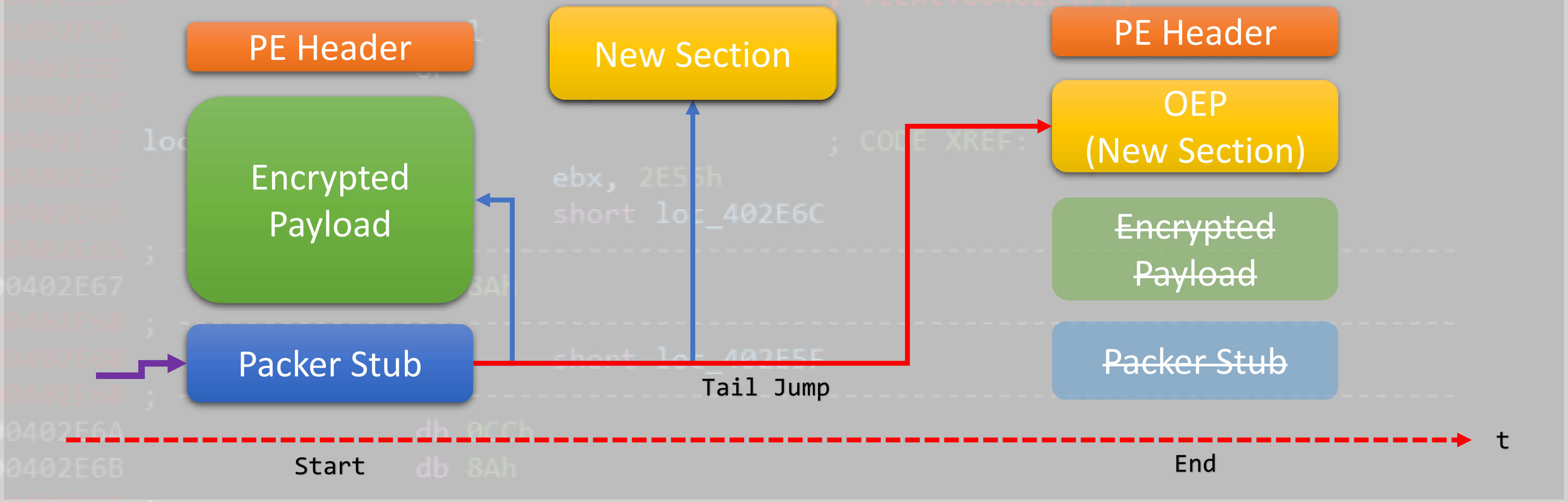
# Code substitution packers

Replace some parts of the original executable mapped into memory by the OS loader.



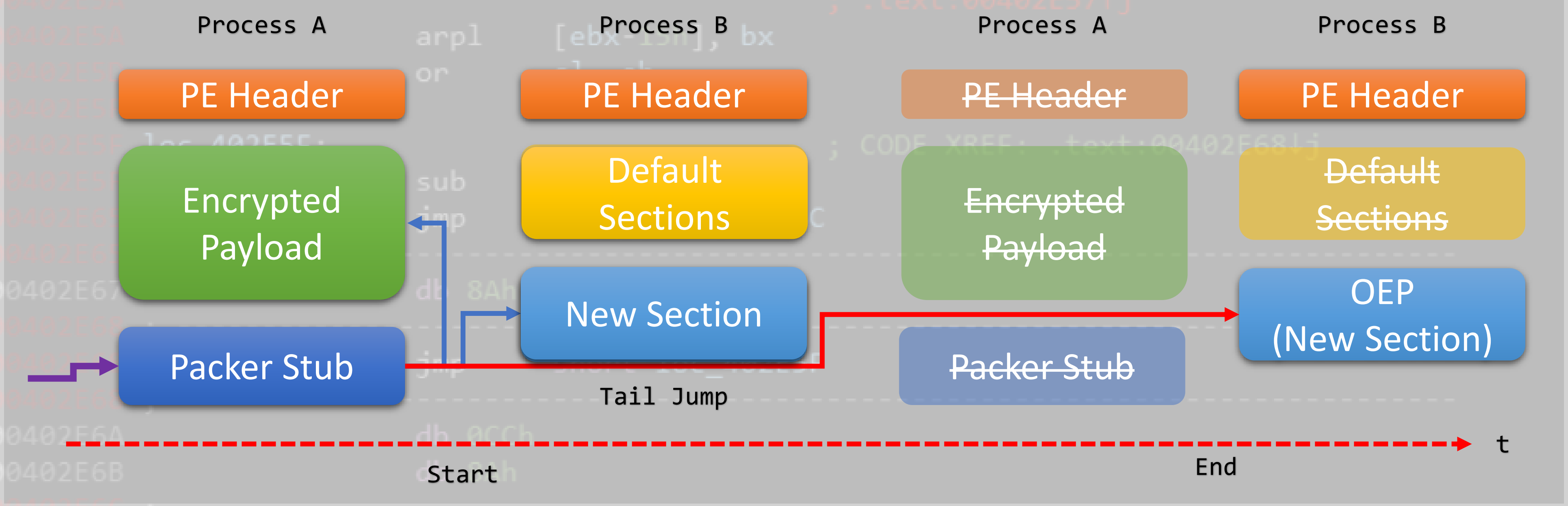
# Code injection packers – Self injection

Allocate new memory sections in the same process and writes shellcode or complete PE files to execute.



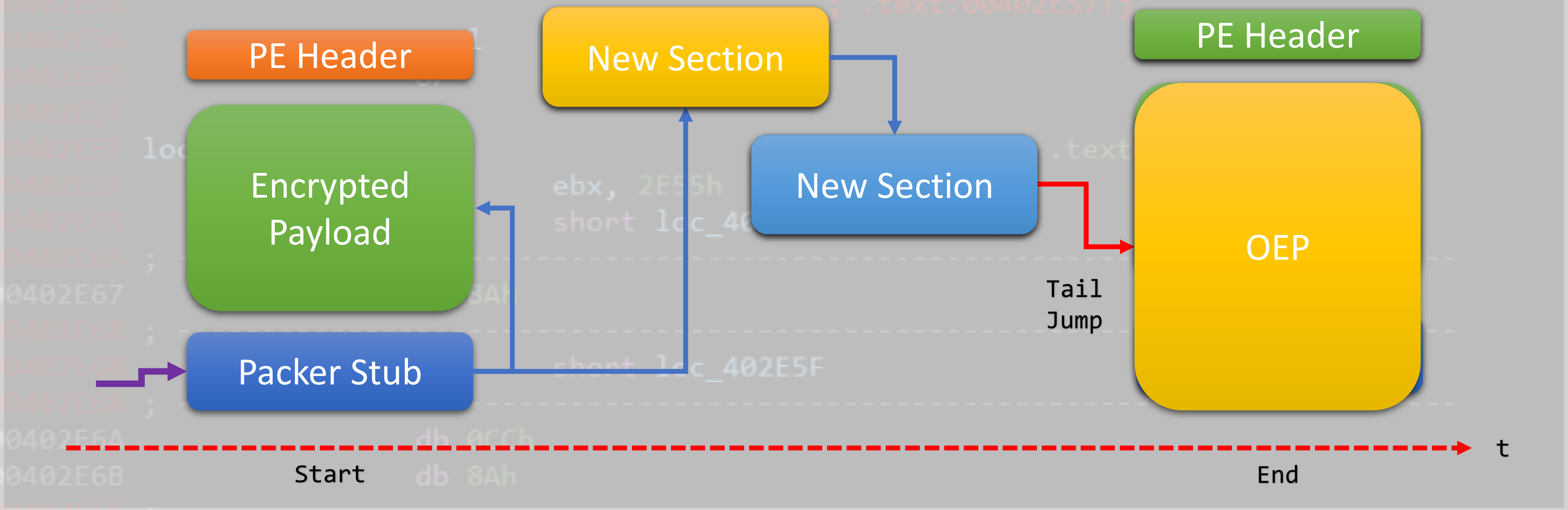
# Code injection packers – Process injection

Allocate new memory sections in external processes and writes shellcode or complete PE files to execute.



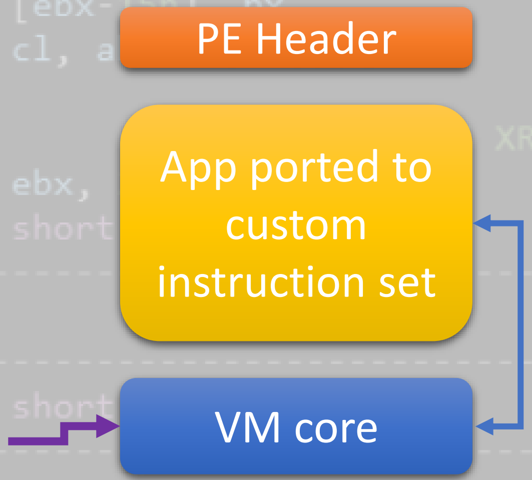
# Hybrid packers – Injection & Substitution

Allocate new memory sections in the same process and writes shellcode or complete PE files to execute.



# Code virtualization packers

Contains a virtual machine and a copy of the program ported to a custom set of instructions (only known by the VM) that are interpreted in run-time.



# Hints

If you are dealing with code substitution, self injection or hybrid packers, you should start monitoring the following Windows APIs:

1 ***VirtualProtect****:* It changes the protection on a region of committed pages in the virtual address space of the calling process. Usually, it is found close to the **Tail Jump**.

2 ***VirtualAlloc****:* Reserves, commits, or changes the state of a region of pages in the virtual address space of the calling process. Memory allocated by this function is automatically initialized to zero.

3 ***LocalAlloc*** *and* ***GlobalAlloc****:* Allocates the specified number of bytes from the heap.

# Hints

If you are dealing with process injection packers, Windows API calls such as CreateProcessA, WriteProcessMemory, VirtualAllocEx and ResumeThread are just the tip of the iceberg.

Study the interaction of each API with the OS for each type of injection is important if you want to master this.

References:

- Ten process injection techniques by elastic.
- ATT&CK Technique T1055 by MITRE.

Hints

These are not the only Windows APIs; however, they are a good starting point. If you are having troubles with a sample, just keep digging!

Thank you!

We are always looking for more talent, if you think you have the skills to join the team, feel free to contact me.

Felipe Duarte
Email: feliped@securityjoes.com
Twitter: @dark0pcodes