



WAVESTONE

Uncommon Process Injection Pattern

Process injection without the standard Win32API calls

07/01/2023





/ **01**

Whoami

And why should we trust you ?



Muggle identity

- › Yoann DEQUEKER (*@OtterHacker*)
- › 26 yo
- › Personal website: *otterhacker.github.io*
- › OSCP, Cybernetics ...



Experience

- › Senior pentester *@Wavestone* for almost 4 years
- › Dedicated to large-scale *RedTeam* operation – *CAC40* companies
- › Development of internal tooling – Mainly malware and Cobalt
- › Malware development workshop *@Defcon31*



/ **02**

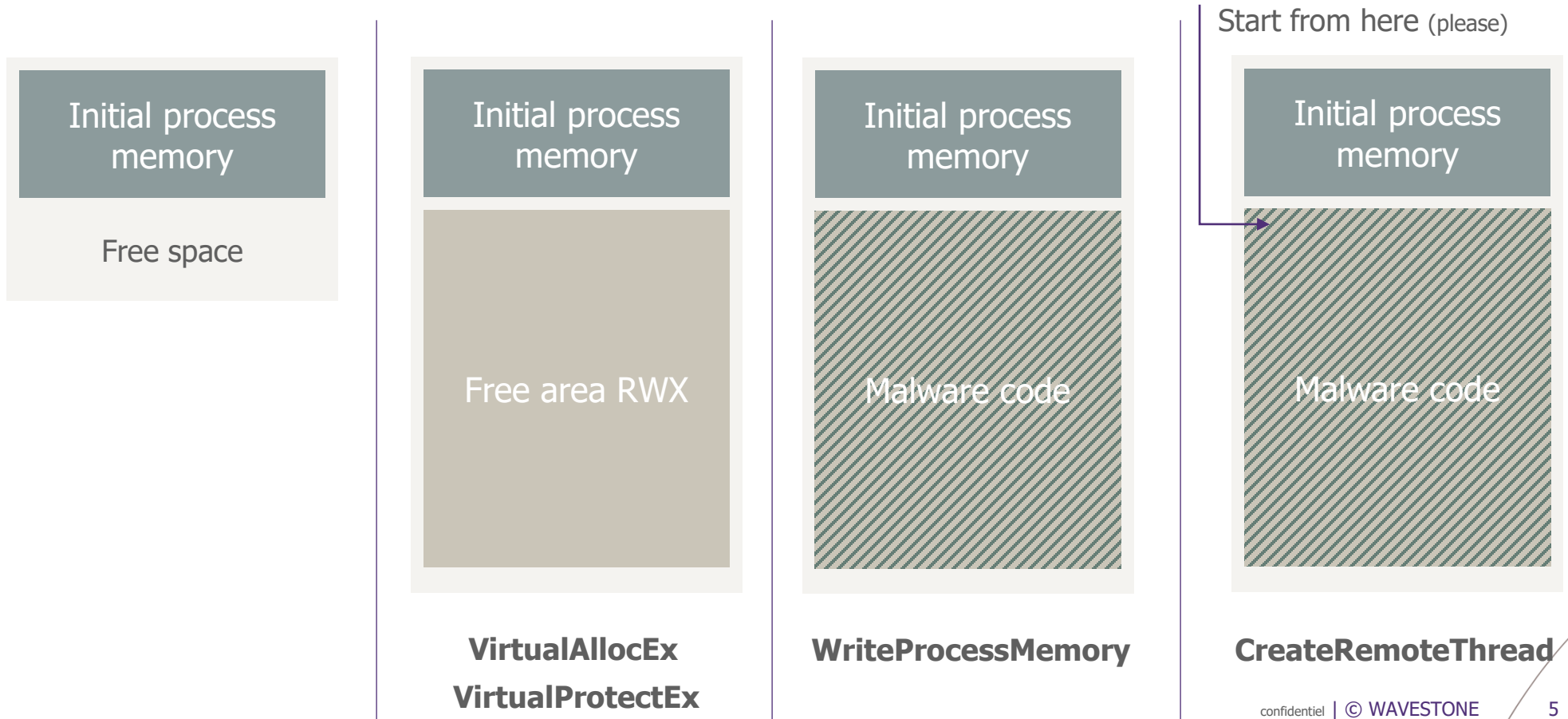
Introduction

Process injection 101

> *Standard pattern*

Main idea

- > Modify the memory of an existing process to inject a malicious binary code
- > Compel the injected process to run the malicious code



Process injection 101

> *Standard pattern*

Main idea

- > Modify the memory
- > Compel the injecte

Initial process
memory

Free space



t from here (please)

Initial process
memory

Malware code

VirtualProtectEx

createRemoteThread

What you will learn today

> *Unusual process injection patterns*



Allocation primitives

- > Drawback of `VirtualAlloc`
- > `LoadLibrary` and `ModuleStomping`



Execution primitives

- > Redirecting execution flows without `CreateRemoteThread`
- > Adaptation of `ThreadLess` injection (by *EthicalChaos*)



Detection mechanism

- > *EDR* hooking basics
- > Fight against *EDR* hooks with a self-debugging code (by *rad9800*)

These techniques have been found by other malware developers, I just adapted them...



/ **03**

Next stopover : VirtualAllocEx land

Allocation primitives: VirtualAllocEx

> System backed and unbacked memory

Effect of VirtualAllocEx

- > The memory space allocated is not recognized to have any use by the system
- > Maybe you should directly send a mail to the SOC...

0x7ff87adb1000	Image: Commit	180 kB	RX	C:\Windows\System32\shlwapi.dll
0x7ff87ae10000	Private: Commit	4 kB	RX	
0x7ff87ae21000	Image: Commit	580 kB	RX	C:\Windows\System32\user32.dll
0x7ff87afd1000	Image: Commit	412 kB	RX	C:\Windows\System32\advapi32.dll
0x7ff87b0f1000	Image: Commit	120 kB	RX	C:\Windows\System32\imm32.dll

Effect of LoadLibraryA

- > A memory space is allocated and backed by the system
- > The memory space is known to have a real purpose

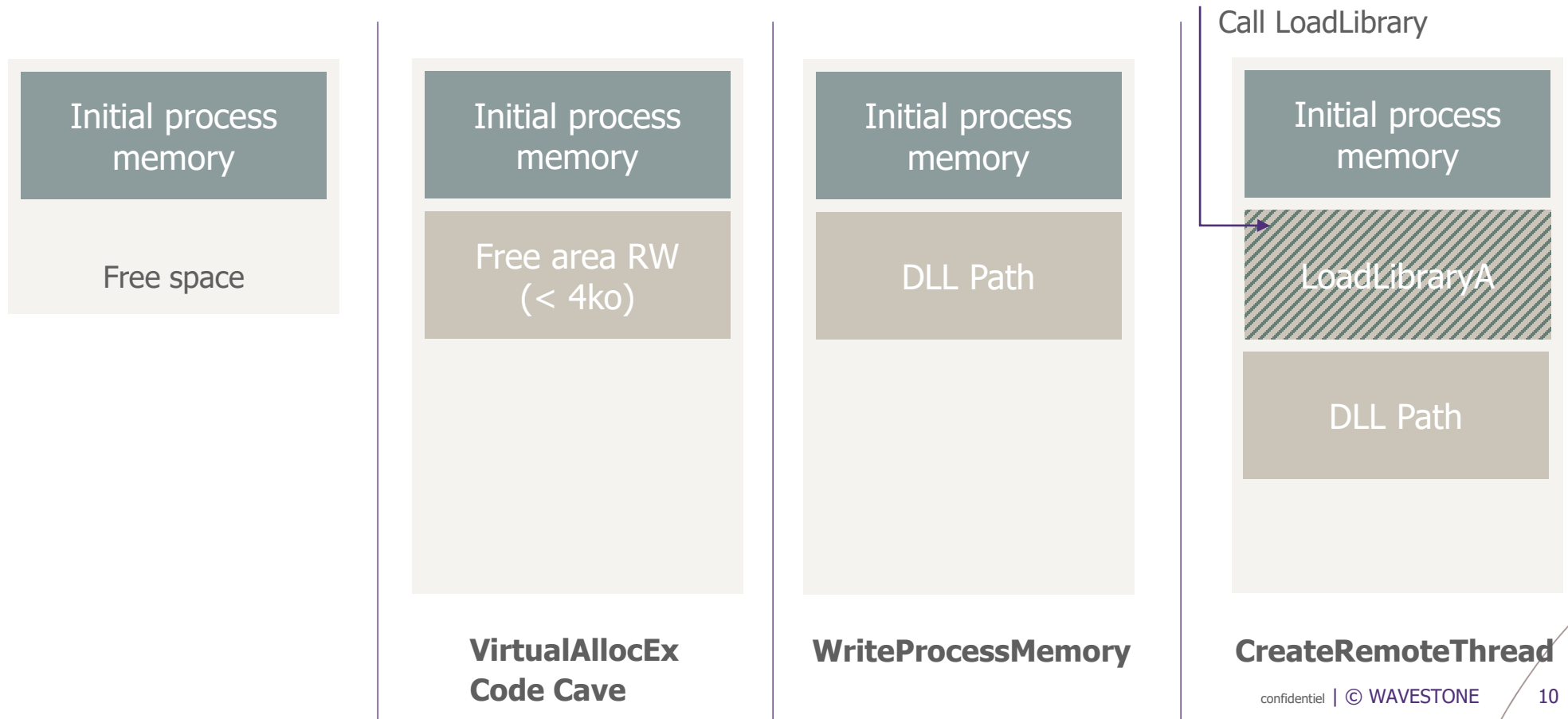
0x7fffe6de0000	Image: Commit	4 kB	R	C:\Windows\System32\winmde.dll
0x7fffe6de1000	Image: Commit	1,372 kB	RX	C:\Windows\System32\winmde.dll
0x7fffe6f38000	Image: Commit	224 kB	R	C:\Windows\System32\winmde.dll
0x7fffe6f70000	Image: Commit	56 kB	RW	C:\Windows\System32\winmde.dll
0x7fffe6f7e000	Image: Commit	72 kB	R	C:\Windows\System32\winmde.dll

How to do it ?

> Use VirtualAllocEx to avoid VirtualAllocEx

VirtualAllocEx again ?

- > EDR does not seem to be bothered by allocation of less than 4ko



What's next with it ?

> Limit the use of VirtualProtect by reusing DLL sections

Reuse the DLL sections ...

- > DLL have predefined sections with specific RWX rights
- > It is interesting to write your malware on the .text section

... And be carefull

- > When writing the remote process, make sure to stay in the .text section
- > Check if there is enough space to write in the DLLMain
- > Use JMP shellcode otherwise







Synthesis (1/2)

> What does an EDR say about it ?

Detection with VirtualAllocEx

- > Detection of anomalous memory detection
- > Detection of code execution from an unbacked memory area

<input type="checkbox"/>	Mar 31, 2023 1:13:51.452 PM		 Anomalous memory allocation in notepad.exe process memory
<input type="checkbox"/>	Mar 31, 2023 1:13:51.452 PM		 Anomalous memory allocation in notepad.exe process memory

Detection with Module Stomping

- > The memory allocated does not rise any specific alerts
- > The code is executed from a backed memory area

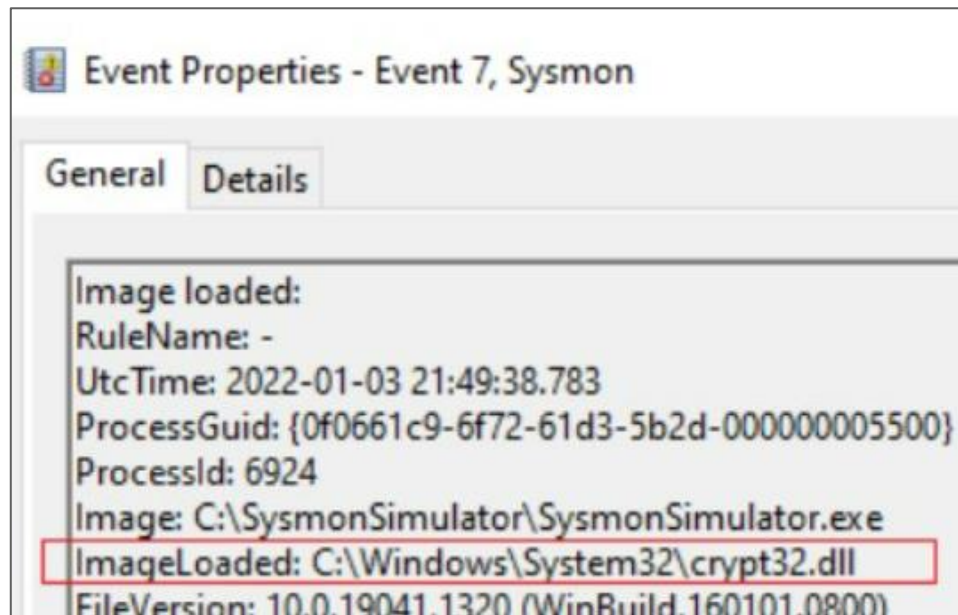
Synthesis (2/2)

> What does an EDR say about it ?

IOC

- > *LoadLibraryA* still raises an *ETW* event that can be caught
- > Heavy use of *CreateRemoteThread*

<input type="checkbox"/>	Mar 28, 2023 6:06:33.048 PM			StompLoader_ntdll.exe created a thread remotely inside notepad.exe	
<input type="checkbox"/>	Mar 28, 2023 6:06:33.048 PM			stomploader_ntdll.exe injected to notepad.exe process	





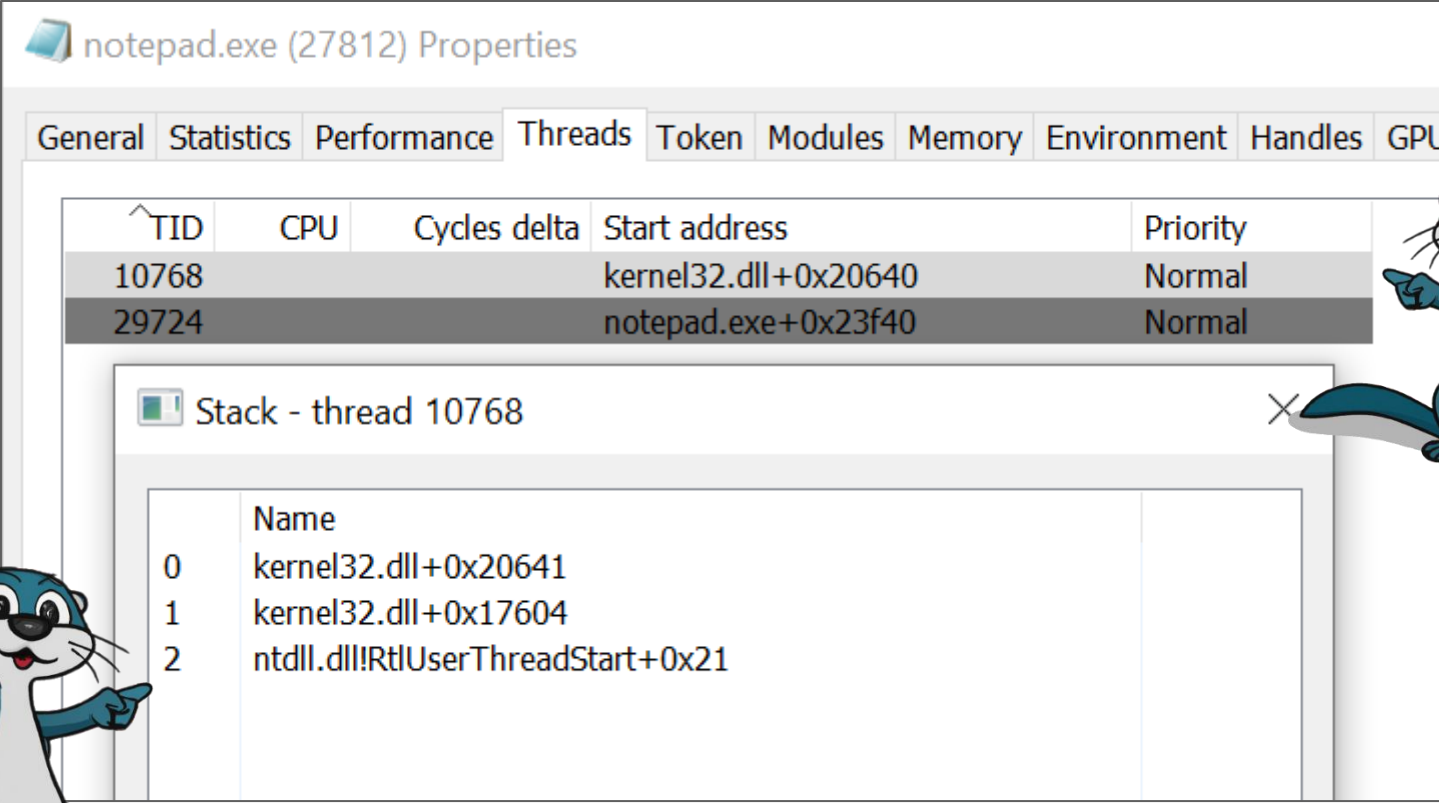
/ **04** Hijack execution flow : CreateRemoteThread

Execution primitives: *CreateRemoteThread*

> Thread and threadless

Effect of CreateRemoteThread

- > CreateRemoteThread is exclusively used to compel the process to execute code at a given start address



notepad.exe (27812) Properties

General Statistics Performance **Threads** Token Modules Memory Environment Handles GPU

TID	CPU	Cycles delta	Start address	Priority
10768			kernel32.dll+0x20640	Normal
29724			notepad.exe+0x23f40	Normal

Stack - thread 10768

	Name
0	kernel32.dll+0x20641
1	kernel32.dll+0x17604
2	ntdll.dll!RtlUserThreadStart+0x21

Seems legit AF

What a nice IOC here

Execution primitives: *CreateRemoteThread* > Thread and threadless (2)

Threadless injection

- > The goal is to compel the program to execute a given code
- > Instead of relying on the *CreateRemoteThread*, we will just wait for the injected process to run the malicious code



Execution primitives: *CreateRemoteThread*

> Thread and threadless (3)

Threadless injection

- > The goal is to compel the program to execute a given code
- > Instead of relying on the *CreateRemoteThread*, we will just wait for the injected process to run the malicious code
- > Just kidding, I don't like to wait



Execution primitives: *CreateRemoteThread*

> A little push up



Notepad.exe

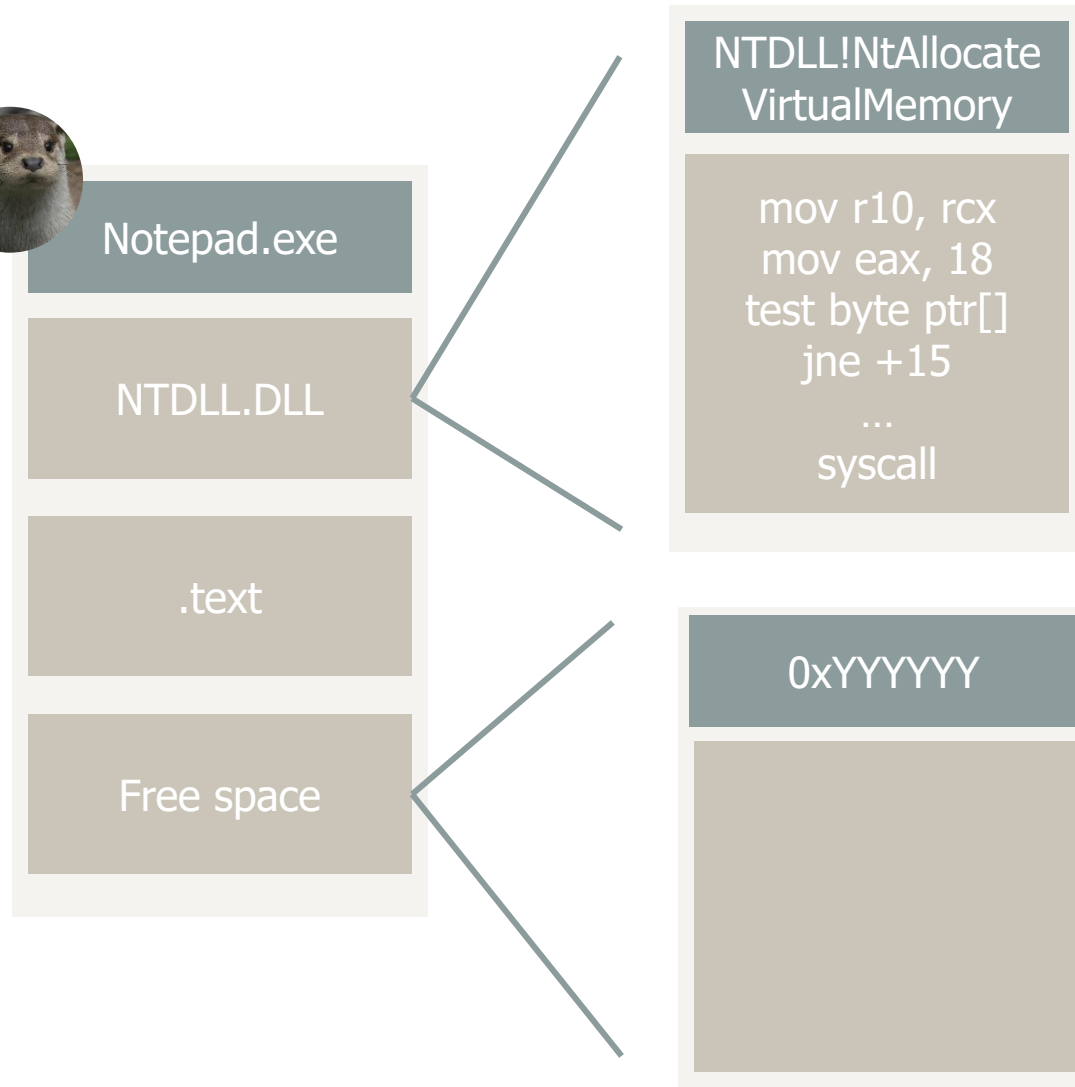
NTDLL.DLL

.text

Free space

Execution primitives: *CreateRemoteThread*

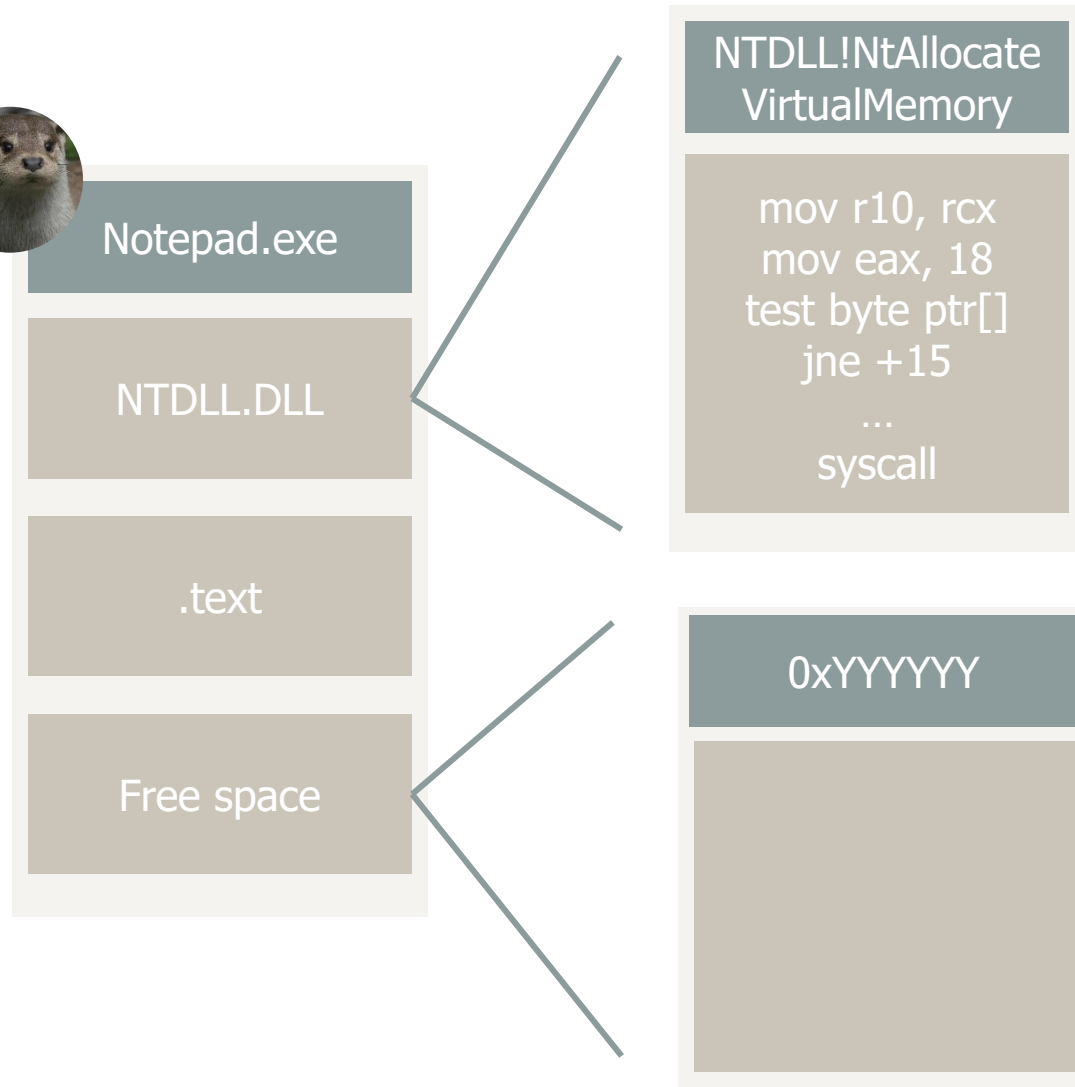
> A little push up



This is the original code of NtAllocateVirtualMemory. Any function that is likely to be called by the injected process will work

Execution primitives: *CreateRemoteThread*

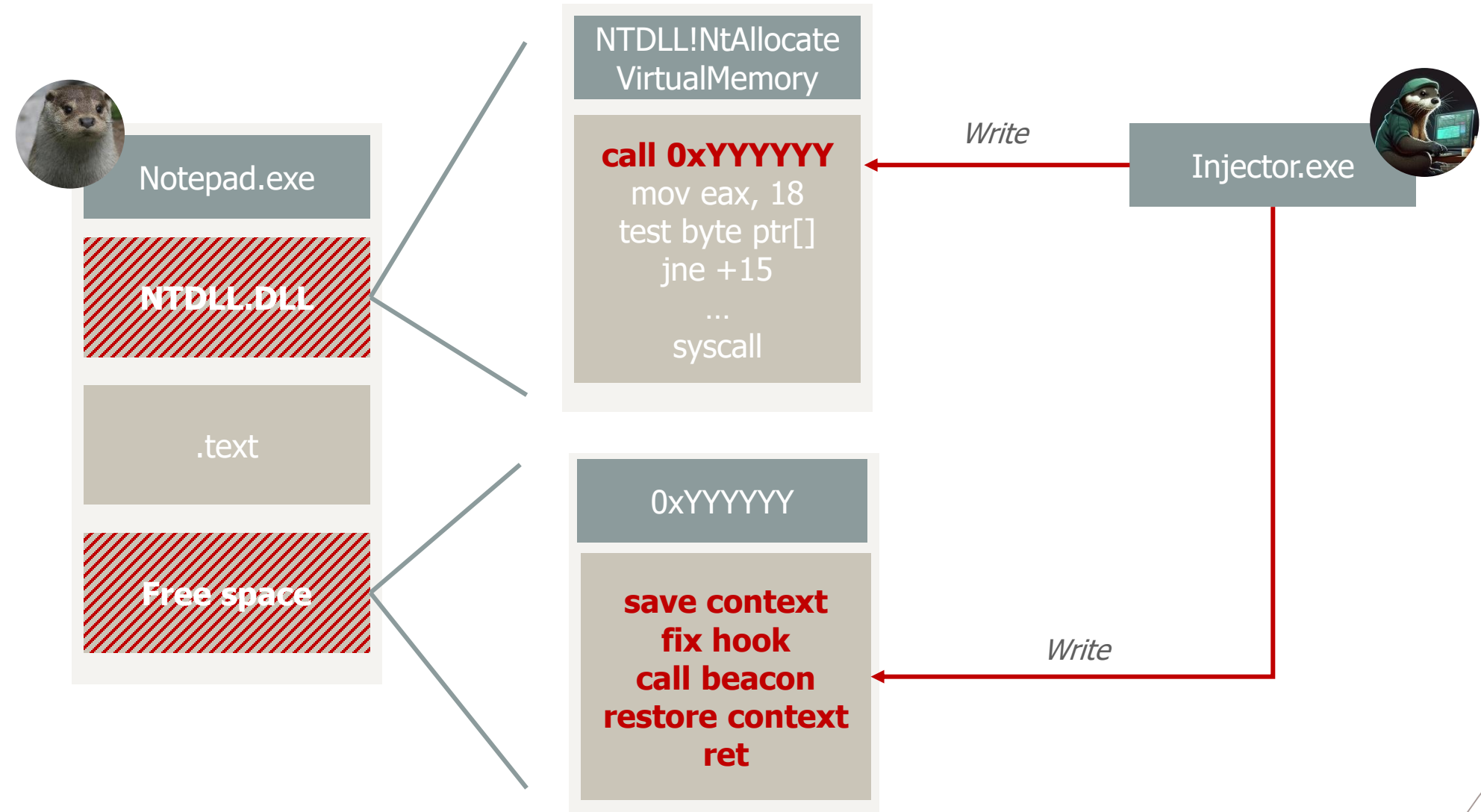
> A little push up



*This is a code cave.
Can also be created with
VirtualAlloc if less than
4ko to limit detection of
anomalous memory
allocation*

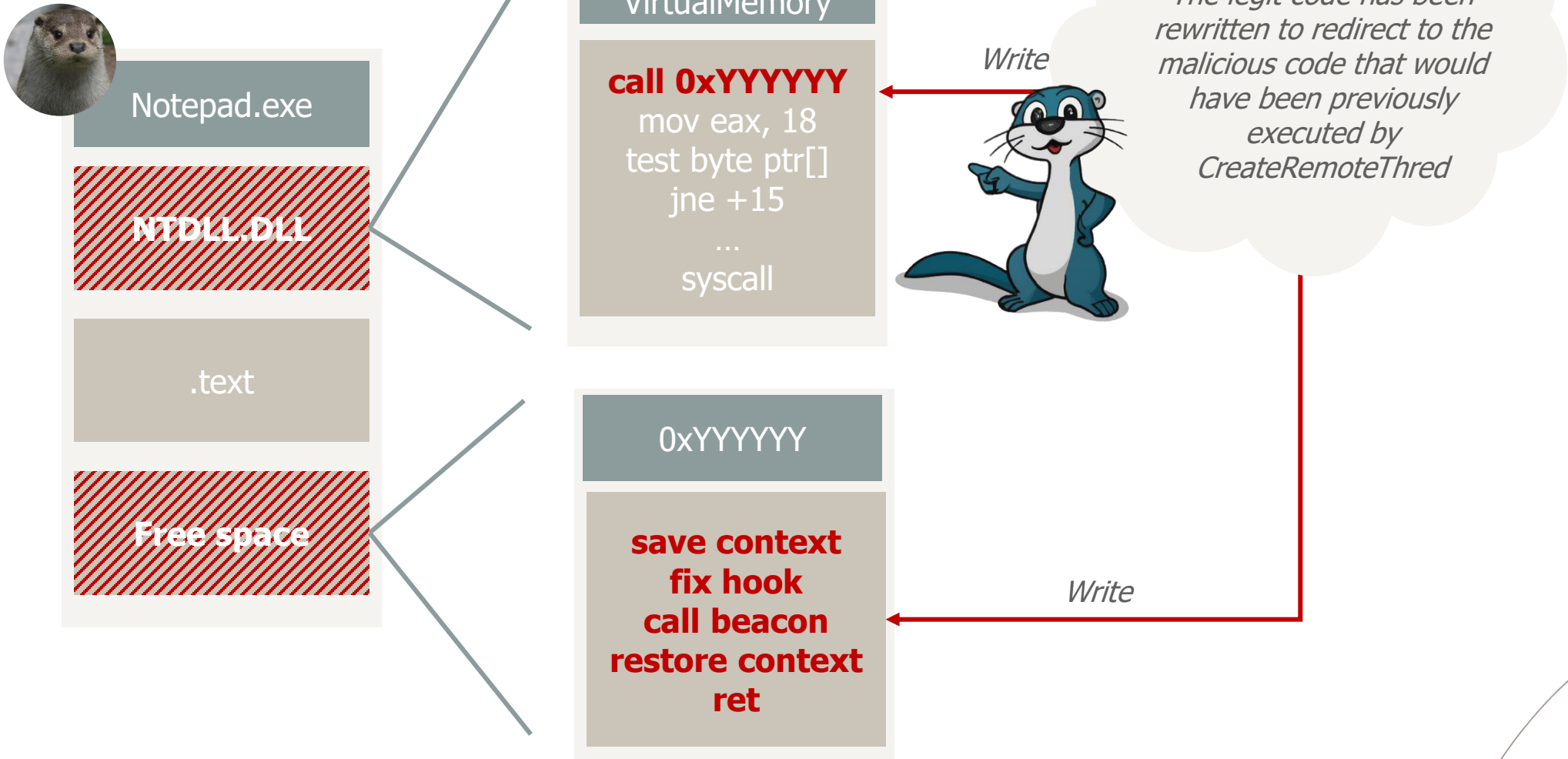
Execution primitives: *CreateRemoteThread*

> A little push up



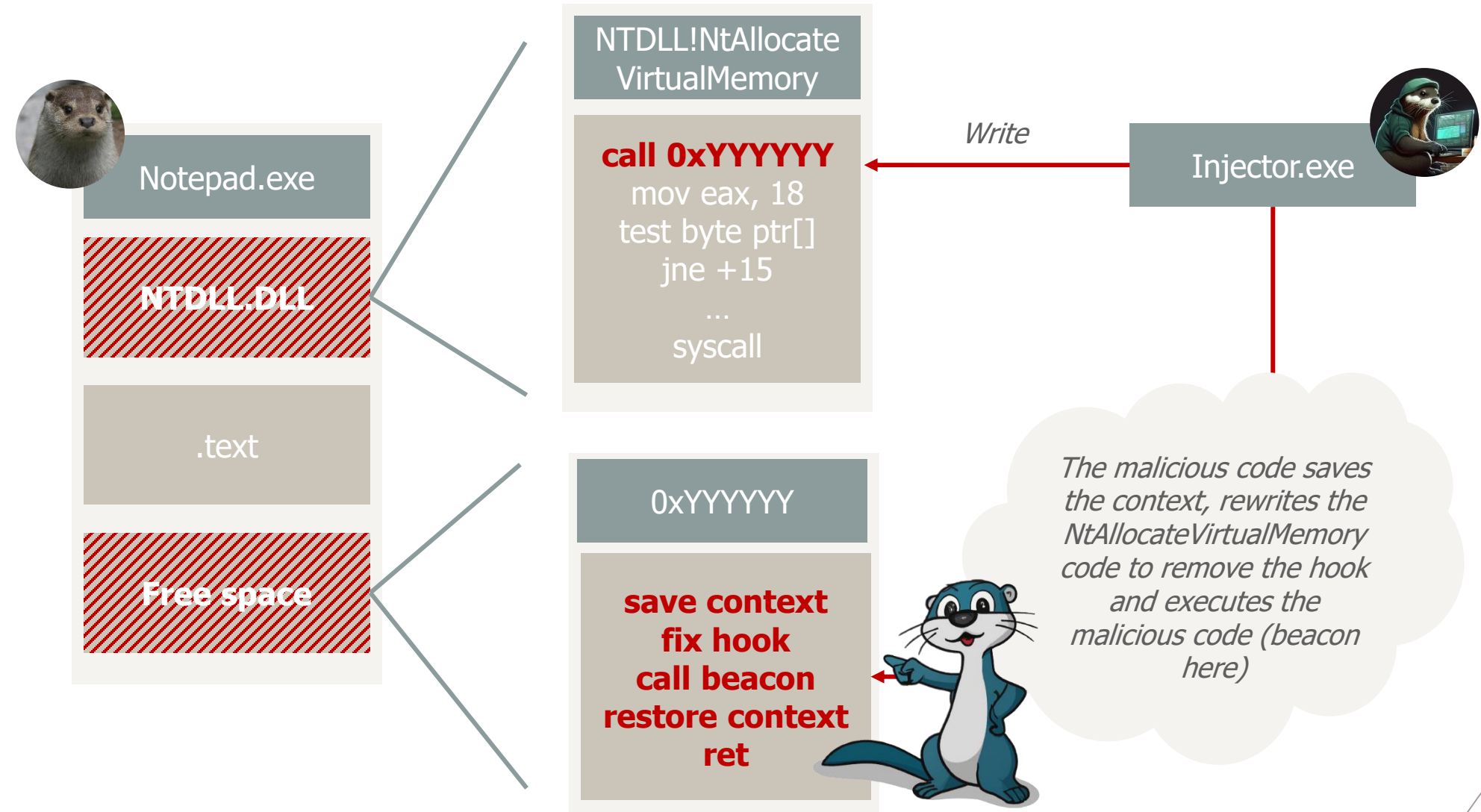
Execution primitives: *CreateRemoteThread*

> A little push up



Execution primitives: *CreateRemoteThread*

> A little push up



Execution primitives: ThreadLess injection

> Thread and threadless

Effect of the ThreadLess injection

- > The malicious code has been successfully executed without using CreateRemoteThred

*Yey ! No
RtlUserThread
in the callstack*

A screenshot of a Windows task manager window. The title bar is purple and shows 'notepad.exe' and the PID '6028'. Below the title bar is a window titled 'Stack - thread 6580'. Inside this window is a table with 4 rows. The first row has a blue header 'Name'. The subsequent rows are indexed 0 to 3. The first row (index 0) shows 'KernelBase.dll!LoadLibraryW'. The other rows show memory addresses: '0x1e2b206001e', '0xa000000078', and '0xccc41fb230'.

	Name
0	KernelBase.dll!LoadLibraryW
1	0x1e2b206001e
2	0xa000000078
3	0xccc41fb230







*Bruuuuu... You've f
the thread stack* confidentiel | © WAVESTONE

Synthesis (1/2)

> What does an EDR say about it ?

Detection with ThreadLess injection

- > The EDR does not detect the injection
- > No complaint about creation of remote thread

<input type="checkbox"/>	Apr 3, 2023 10:16:57.330 AM		 notepad.exe established connection with 10.253.0.3:80
<input type="checkbox"/>	Apr 3, 2023 10:16:28.402 AM		 User SRV02\Administrator launched process notepad.exe

T1204: User Execution

Synthesis (1/2)

> What does an EDR say about it ?

Detection with ThreadLess injection

- > The EDR does not detect the injection
- > No complaint about creation of remote thread

- | | | | |
|--------------------------|-----------------------------|---|--|
| <input type="checkbox"/> | Apr 3, 2023 10:16:57.330 AM | 🚩 | 🌐 notepad.exe established connection with 10.253.0.3:80 |
| <input type="checkbox"/> | Apr 3, 2023 10:16:28.402 AM | 🚩 | ⚙️ User SRV02\Administrator launched process notepad.exe |

T1204: User Execution

- | |
|---|
| ⚙️ StompLoader3.exe changed the protection of a memory region in the address... |
| ⚙️ StompLoader3.exe changed the protection of a memory region in the address... |
| ⚙️ StompLoader3.exe changed the protection of a memory region in the address... |
| ⚙️ StompLoader3.exe changed the protection of a memory region in the address... |
| ⚙️ StompLoader3.exe changed the protection of a memory region in the address... |

Synthesis (1/2)

> Is it bulletproof ?



RWX protection on hooked function

- › Use of RWX on hooked function to allow the hook to restore the original code
- › The hook function can perform the *VirtualProtect* call by itself
- › Will increase the hook size, therefore the possible detection



Unclean threadstack and shellcode

- › The call of some function can mess with the thread call stack (*LoadLibrary* for example)
- › The call stack will show jump to unusual memory addresses
- › Use of hardware breakpoint to avoid directly patching the remote process



EDR hooks

- › The injection is still sensible to *EDR* hooks
- › The injector can still be flagged as malicious once the injection ended
- › Bypassing EDR hooks can be a nice addition



/ **05**

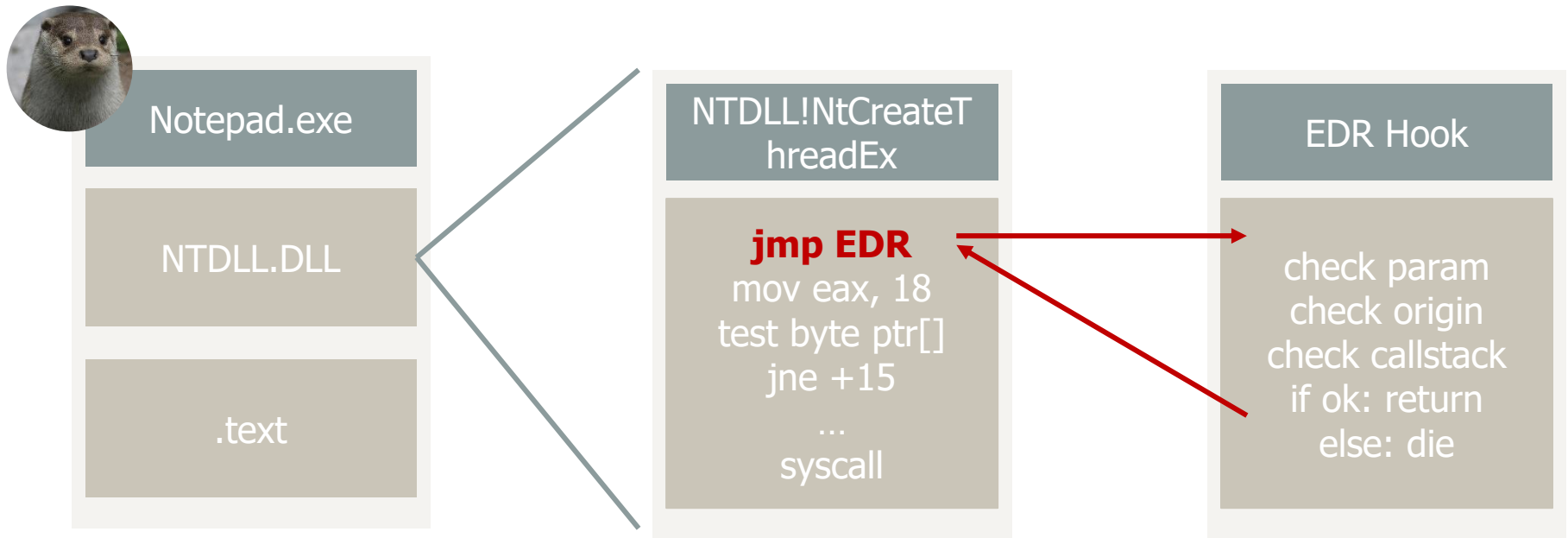
Nothing to see here : bypassing userland
hooks

EDR hooks 101

> Hooks, Userland and KernelLand

Interest of EDR hooks

- > Placing hooks on sensitive functions such as CreateRemoteThread or NtAllocateVirtualMemory allows the EDR to prevent their execution

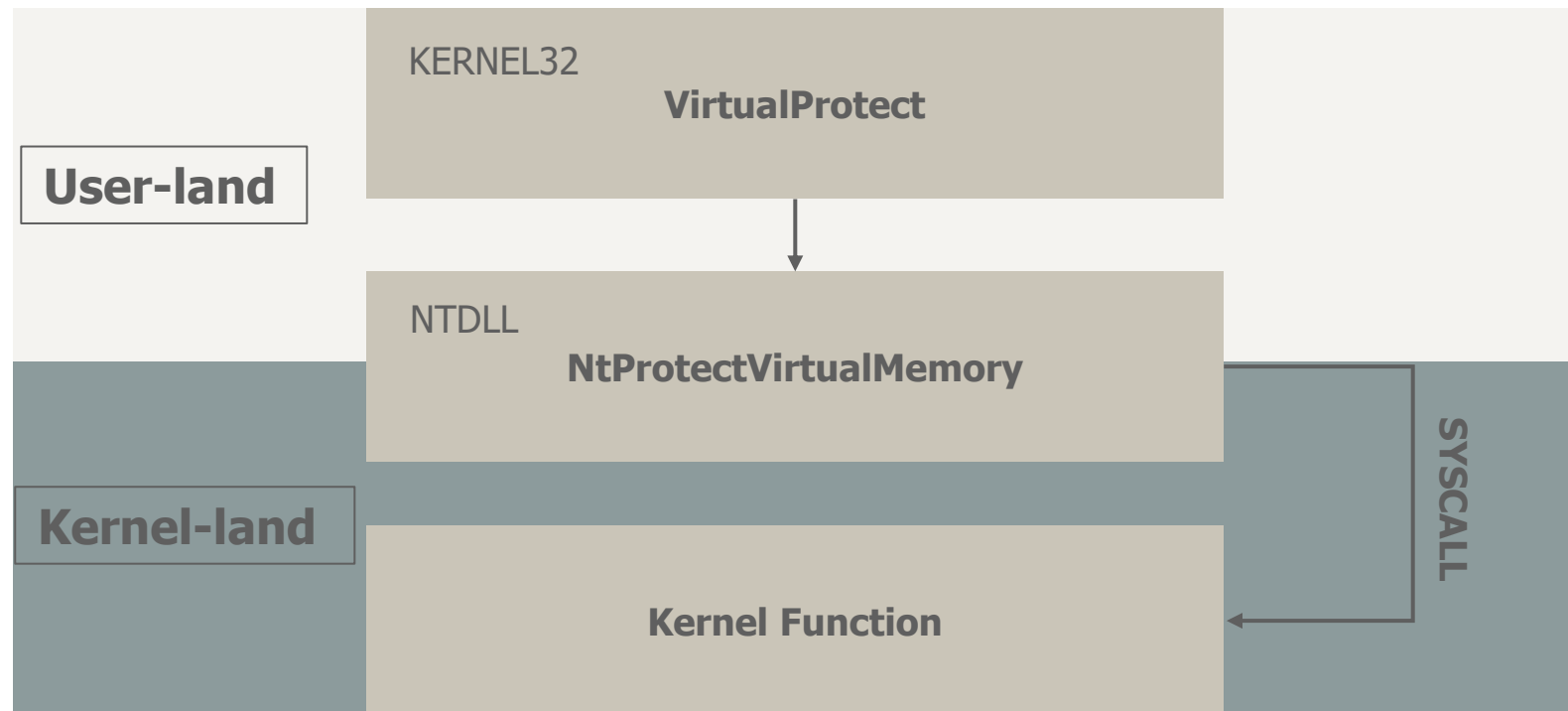


EDR hooks 101

> Hooks, Userland and KernelLand

Userland VS KernelLand

- > EDR can easily inject hooks on userland function to **prevent** their use
- > EDR can use kernel callbacks to detect **use** of sensitive functions



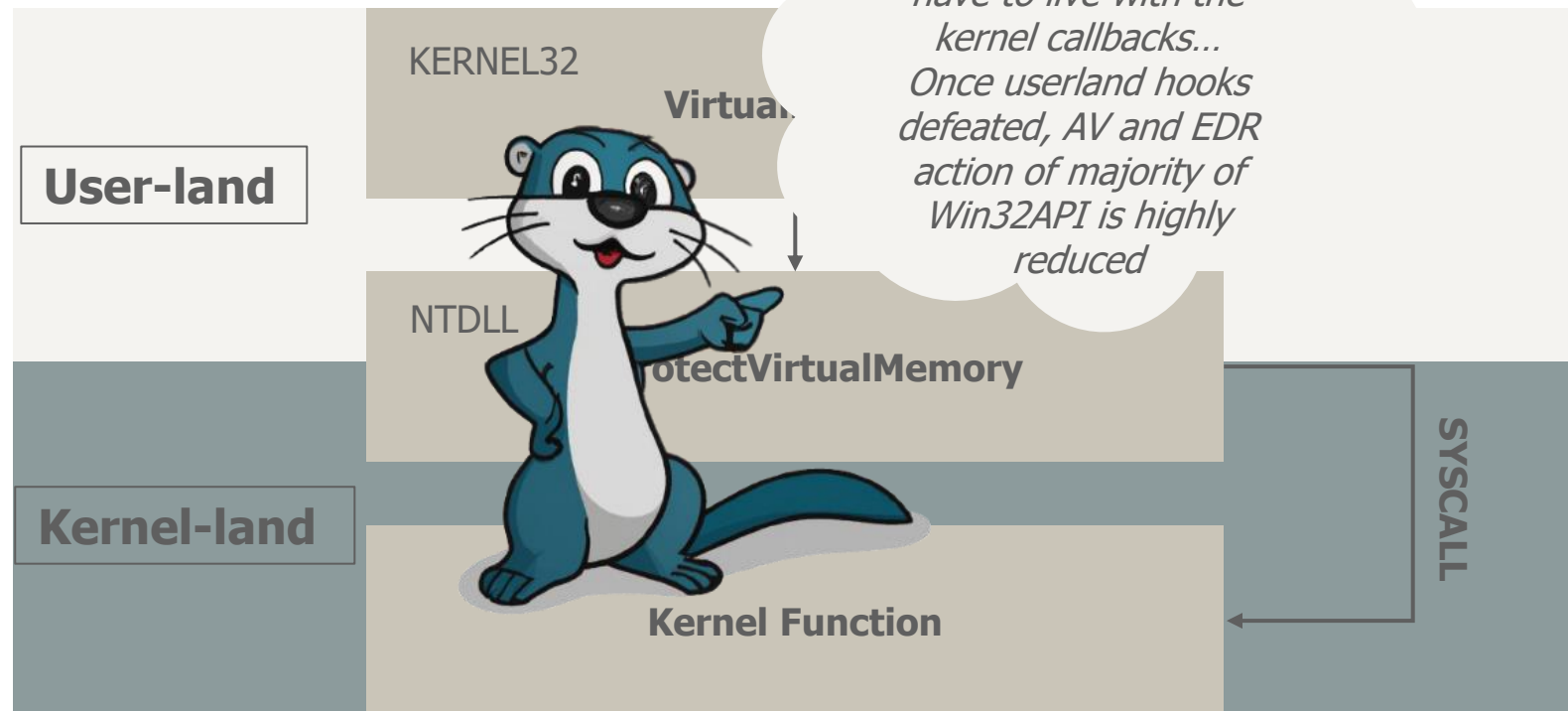
EDR hooks 101

> Hooks, Userland and KernelLand

Userland VS KernelLand

- > EDR can easily inject hooks on userland function to **prevent** their use
- > EDR can use kernel callbacks to detect **use** of sensitive f

Userland hooks can be easily removed, but we have to live with the kernel callbacks... Once userland hooks defeated, AV and EDR action of majority of Win32API is highly reduced



Bypass userland hooks

> Patching vs debugging

Patching

- › Detect the EDR hook in the function and replace it
- › Can trigger EDR integrity check

Bypass userland hooks

> Patching vs debugging

Patching

- > Detect the EDR hook in the function and replace it
- > Can trigger EDR integrity check

*Patching the EDR hook
implies the use of
VirtualProtect that can
also be hooked...*

*Even if it seems to be the
simplest approach, it
might not be the best*



Bypass userland hooks

> Patching vs debugging

Patching

- › Detect the EDR hook in the function and replace it
- › Can trigger EDR integrity check

Hardware breakpoint

- › Set a breakpoint on the syscall instruction
- › Call the function with random parameter
- › Wait for the breakpoint to be triggered
- › Replace the random parameters in the stack
- › Continue the execution

Bypass userland hooks

> Patching vs debugging

Patching

- > Detect the EDR hook in the function and replace it
- > Can trigger EDR integrity check

This is not a dehooking technique.

The EDR hook is neither modified nor deleted.

Hardware breakpoint

- > Set a breakpoint on the syscall instruction
- > Call the function with random parameters
- > Wait for the breakpoint to be triggered
- > Replace the random parameters in the registers
- > Continue the execution

The breakpoint allows the modification of the syscall parameters just in time



Bypass userland hooking

> Debugging



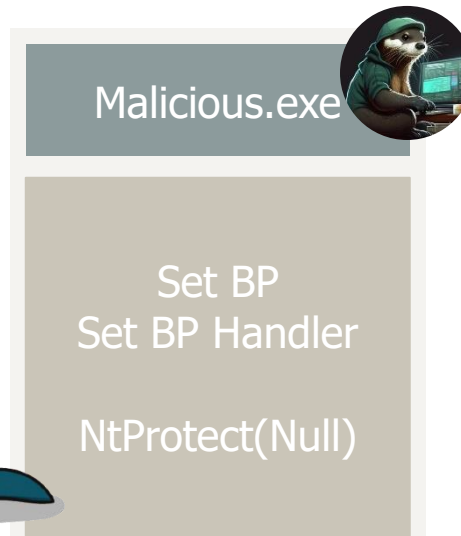
*A breakpoint is set to be triggered when the **SYSCALL** instruction is going to be executed. This is done by setting the **Dr0, Dr7 and Dr6** context registers*

Bypass userland hooking

> Debugging

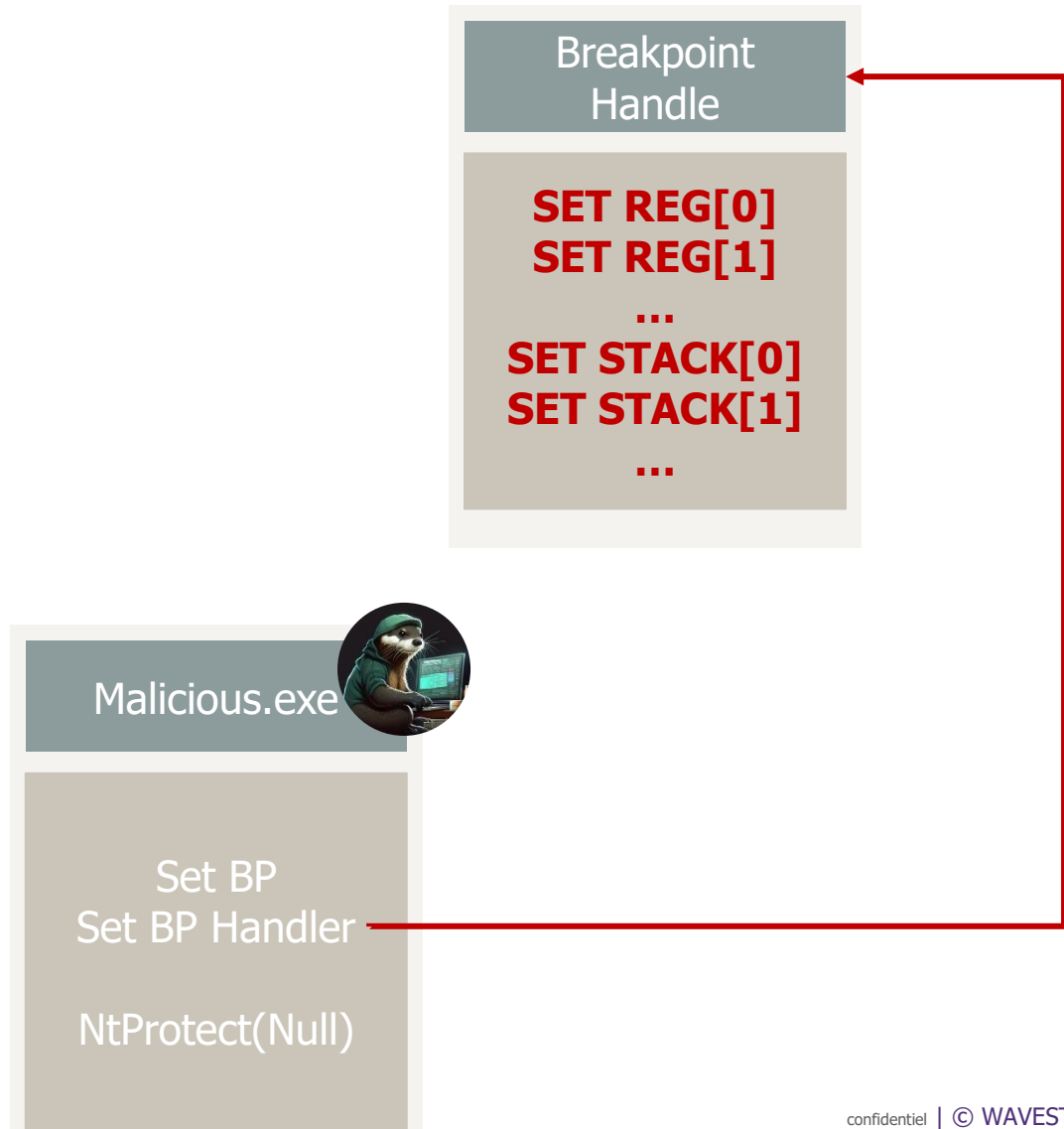
*A breakpoint handler is registered using the **SetUnhandleException Filter** function.*

Any exception not handled by the code will be processed by the defined handler



Bypass userland hooking

> Debugging



Bypass userland hooking

> Debugging

*The breakpoint handler
modify the registers and
the stack in order to
change the parameter that
will be used by the syscall*



Breakpoint
Handle

**SET REG[0]
SET REG[1]**

...

**SET STACK[0]
SET STACK[1]**

...

Malicious.exe

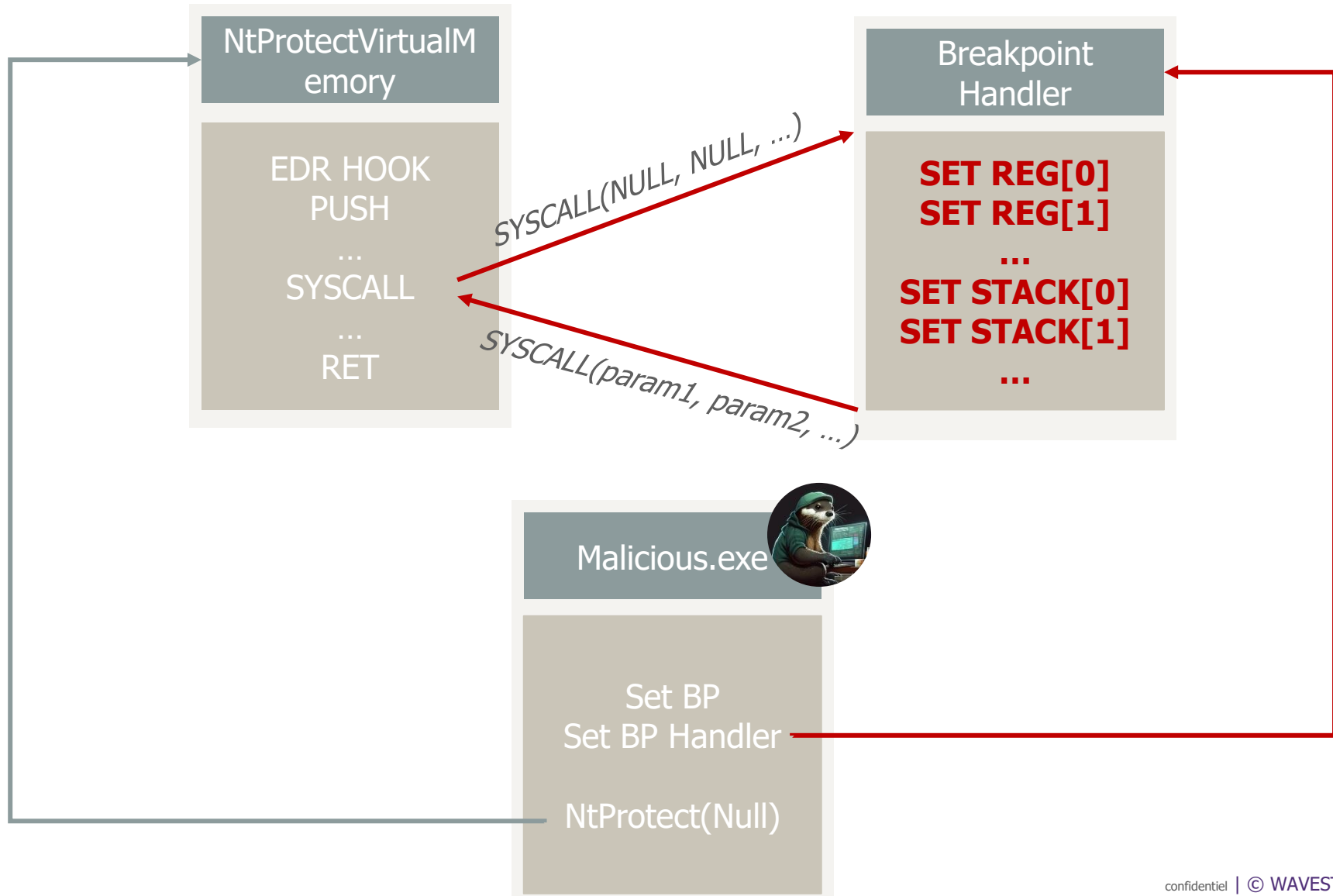


Set BP
Set BP Handler

NtProtect(Null)

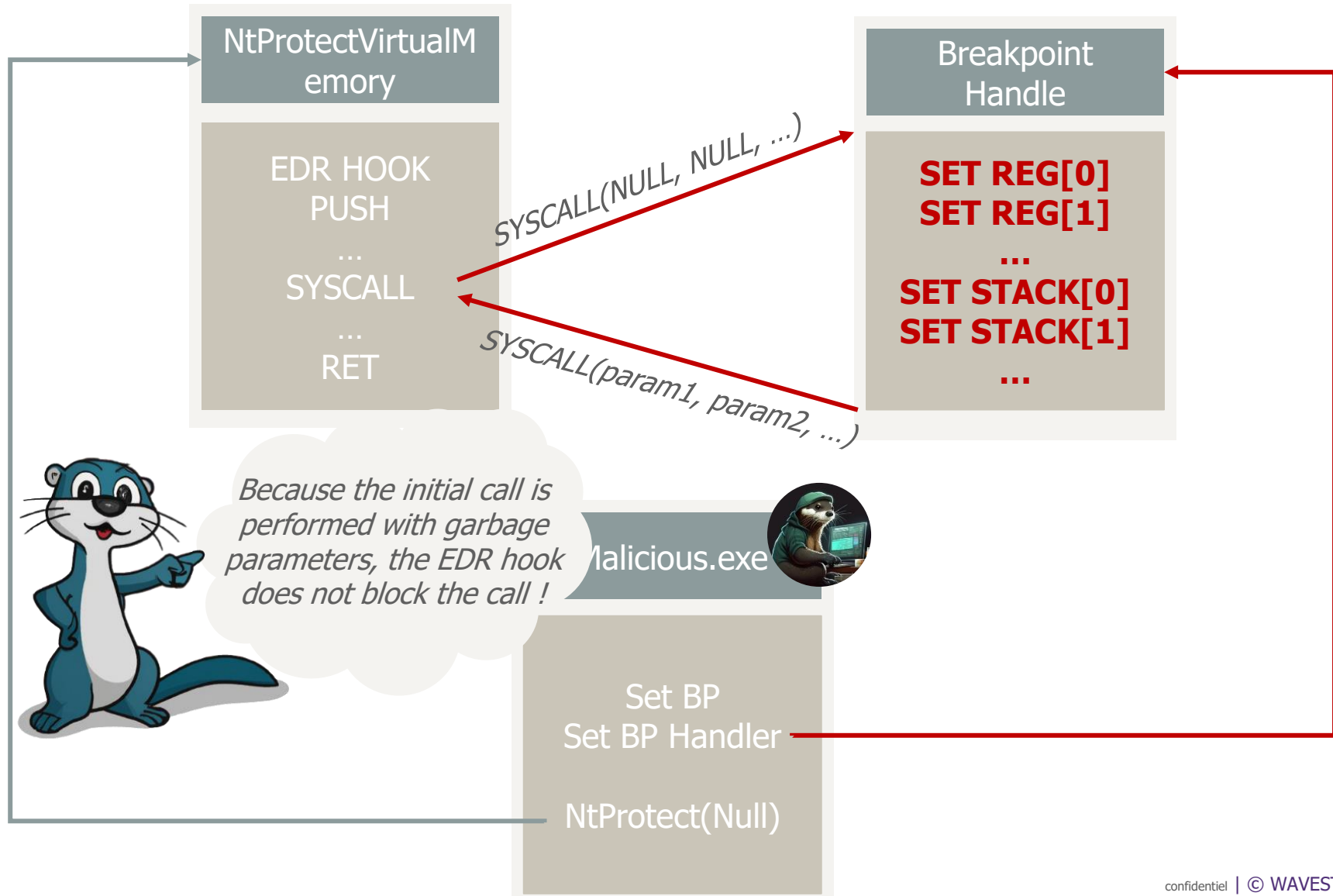
Bypass userland hooking

> Debugging



Bypass userland hooking

> Debugging



QUESTIONS ?



That's all folks ! Thank you !



If you have additional questions, feel free to ask me at the bar

PARIS

LONDRES

NEW YORK

HONG KONG

SINGAPOUR *

DUBAI *

SAO PAULO *

LUXEMBOURG

MADRID *

MILAN *

BRUXELLES

GENEVE

CASABLANCA

ISTANBUL *

LYON

MARSEILLE

NANTES

* Partenariats

WAVESTONE

