Microsoft

MODERN WINDOWS USERSPACE EXPLOITATION

Saar Amar

MSRC-IL

whoami

- @AmarSaar
- Security researcher. Live to reverse, breath to exploit.
- MSRC-IL
- Addicted to CTFs!
  - @pastenctf team member

The reason to live: Exploit!

- We want to execute arbitrary code (duh…)
- Easy!
  - Find an awesome 0day vulnerability
  - ???
  - Profit!
- Yeah…


MITIGATIONS
LIGHTYEAR
MITIGATIONS EVERYWHERE
imgflip.com

Mitigations. Mitigations. Mitigations.

- Control-flow integrity mitigations
  - DEP
  - CFG
  - CFI
- Code integrity mitigations
  - ACG
  - CIG
- Supporting mitigations
  - Child Process Policy
  - ASLR
  - SafeSEH/SEHOP
  - Heap randomization && metadata protection
- Sandboxing / Containers
  - LPAC
  - WDAG

Even more! See https://www.microsoft.com/en-us/msrc/bounty-mitigation-bypass

Exploit through the ages

- aka what is this talk about
- Insomni’hack CTF Teaser as an example vulnerability
- Compare the exploit on different platforms:
  - Windows 7
  - Windows 10 TH1
  - Windows 10 RS5

The “Winworld” challenge

- Awesome challenge by @__awe
- Story based on “Westworld”
- Implements a “narrator” interface, which lets you
  - Create robots and humans
  - Configure their behavior
  - Move them on the map
  - Interact with each other
- Manipulates *Person* objects – a shared class for:
  - Hosts (robots)
  - Guests (humans)

```
--[ Welcome to Winworld, park no 1209 ]--
narrator [day 1]$ help
Available commands:
 - new <type> <sex> <name>
 - clone <id> <new_name>
 - list <hosts|guests>
 - info <id>
 - update <id> <attribute> <value>
 - friend <add|remove> <id 1> <id 2>
 - sentence <add|remove> <id> <sentence>
 - map
 - move <id> {<l|r|u|d>+}
 - random_move
 - next_day
 - help
 - prompt <show|hide>
 - quit
narrator [day 1]$
```

Vulnerability 1: uninitialized attr in Person copy c'tor

- Robots are initialized with *is_conscious*=false in the Person's c'tor
- Person's **copy c'tor** used in the narrator clone function skips this initialization!
- The value will thus be uninitialized
  - use whatever was already on the heap
- By forcing a robot to become a human, we have *is_conscious* uninitialized

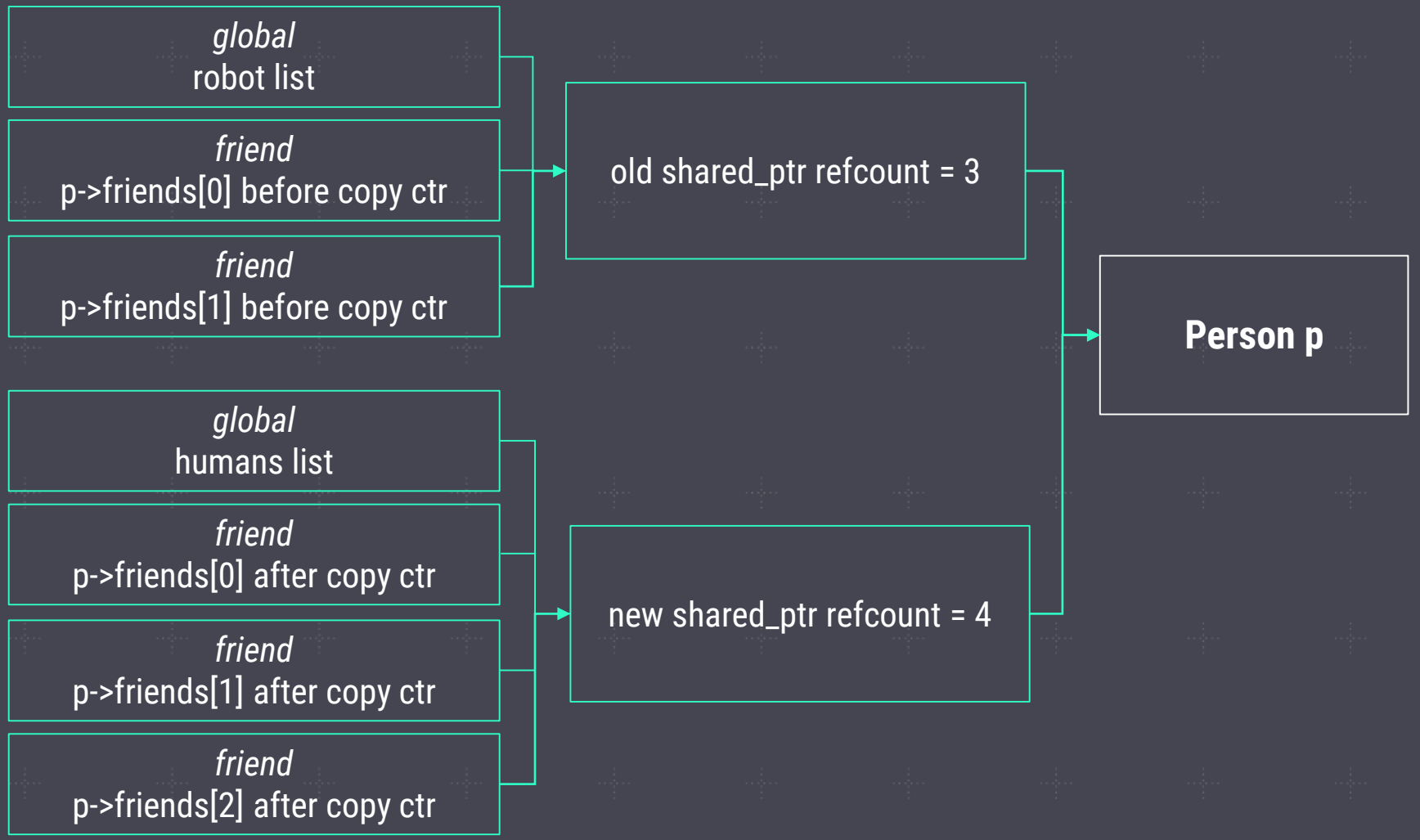
Vulnerability 2: UAF due to misused std::shared_ptr

- When a robot becomes human, it stays in the robots list
  - and gets inserted into the humans list as well!
- Instead of incrementing the refcount of the object's std::shared_ptr, we create a new std::shared_ptr that points to the same object
  - Due to guests.push_back(std::move(p));
- Therefore, when one of the *std::shared_ptr* gets to 0 – we have a dangling pointer!

Vulnerability 2: UAF due to misused std::shared_ptr

```
person->is_enabled = false;
person_t p(person.get());
std::cout << p->getName() << " becomes human!" << std::endl;

p->type = GUEST;

for (auto &host : hosts) {
    if (host.get() == p.get()) {
        continue;
    }

    p->manageFriend(ADD, host);
    host->manageFriend(ADD, p);
}

guests.push_back(std::move(p));
```

Vulnerability 2: UAF due to misused std::shared_ptr



Triggering the clone

- The clone logic is triggered when a robot becomes human
- This happens when a robot reaches the center of the maze, using the move command
- And the following conditions must be met:
  - Currently moved person must be a host (i.e. robot)
  - person->is_conscious is set (!false)
  - There is a guest (i.e. human) in the maze center as well

- For that, we need the uninitialized *is_conscious* vulnerability

LET THE FUN BEGIN

WINDOWS 7

Windows 7 Heap

- Low fragmentation heap (LFH)
- In Windows 7:
  - Chunks in userblocks allocated continuously
  - Different userblocks allocated continuously
  - *FreeEntryOffset* in free chunks content (first 2 bytes)
  - Can redirect next-next allocation to an arbitrary address
- Shaping the LFH is relatively easy
  - Spraying chunks of the same size => continuous allocation
  - *malloc* returns the last freed chunk

Predictable. Everything's predictable

```c
int main(void) {

    HANDLE hHeap = HeapCreate(0, 0, 0);

    printf("[*] activate bucket 0x%x in LFH\n", SIZE);
    spray(hHeap, 0x12, FALSE);

    printf("[*] spray\n");
    spray(hHeap, 0x100, TRUE);

    return 0;
}

void spray(HANDLE hHeap, size_t cnt, BOOL trace) {
    void *p;
    for (size_t i = 0; i < cnt; i++) {
        p = HeapAlloc(hHeap, 0x0, SIZE);
        if (trace) {
            printf("HeapAlloc() == %p\n", p);
        }
    }
}
```

```
c:\share>LFH_pasten.exe
[*] activate bucket 0x100 in LFH
[*] spray
HeapAlloc() == 000000000037C4A0
HeapAlloc() == 000000000037C5B0
HeapAlloc() == 000000000037C6C0
HeapAlloc() == 000000000037C7D0
HeapAlloc() == 000000000037C8E0
HeapAlloc() == 000000000037C9F0
HeapAlloc() == 000000000037CB00
HeapAlloc() == 000000000037CC10
HeapAlloc() == 000000000037CD20
HeapAlloc() == 000000000037CE30
HeapAlloc() == 000000000037CF40
HeapAlloc() == 000000000037D050
HeapAlloc() == 000000000037D160
HeapAlloc() == 000000000037D270
HeapAlloc() == 000000000037D380
HeapAlloc() == 000000000037D490
HeapAlloc() == 000000000037D5A0
HeapAlloc() == 000000000037D6B0
HeapAlloc() == 000000000037D7C0
HeapAlloc() == 000000000037D8D0
HeapAlloc() == 000000000037D9E0
```

Arbitrary Read

- Person->name is an std::string
- *info <id>* prints the name of a human/robot
- Arbitrary read:
  - Corrupt the std::string pointer and size
  - Use *info* to read an arbitrary number of bytes from an arbitrary address

Arbitrary Write

- Person->name is an std::string
- *update <id> name <name>* sets a new name
- Arbitrary write:
  - Corrupt the std::string pointer and size
  - Use *update* to write arbitrary data to an arbitrary address

Code Execution

- Move two persons to the same location on the map
- A function pointer is called on both:
  - this->onEncounter(other)
- Corrupt this function pointer and jump to an arbitrary address

Rule them all

- Corrupt *onEncounter* and set it to *gets*:
  - this->onEncounter(other) becomes gets(other)
- Which means that on every encounter, we can reset an object
- Much easier than exploiting the UAF every time
- I use it for arbitrary reads, mostly

Breaking ASLR

- We need to leak our image base / ntdll / some libraries for gadgets and functions
- Shape the LFH so an *std::vector* object will be allocated on a dangling *Person* after the UAF
- Execute *info* on the dangling *Person*
- Output is the *std::vector* vtable address, from the base image .rdata
- The Ugly: the process dies right after...
  - Libraries VAs are randomized once per boot, and the challenge relaunched ☺

Land of possibilities

- Corrupt vtables / functions pointers
- Corrupt the stack
- Load unsigned DLLs
- Modify or create unsigned code pages:
  - VirtualProtect(…, PAGE_EXECUTE, …)
  - VirtualAllocEx(…, PAGE_EXECUTE, …)
- Everything works!

Windows 7 Exploit

- From here, ROP
- Start with VirtualAllocEx for a new RWX page
- Write shellcode into the new page
- Jump into it
- PROFIT

WINDOWS 7 DEMO

WINDOWS 10 TH1

Shape fails

- Windows 7 version of the exploit fails on Windows 10 TH1
- Turns out the heap shaping fails
- We never get the same allocation as the dangling *Person* object

Heap Randomization and Metadata Protection

NAME

Heap Randomization and Metadata Protection

WORKAROUNDS

- Avoid touching the heap metadata
- Spray allocations against randomization

EXPLAINED

The integrity of heap metadata cannot be subverted and the layout of heap allocations is not predictable to an attacker

BOUNTY

Up to $15K

What the RANDOM!

```c
int main(void) {

    HANDLE hHeap = HeapCreate(0, 0, 0);

    printf("[*] activate bucket 0x%x in LFH\n", SIZE);
    spray(hHeap, 0x12, FALSE);

    printf("[*] spray\n");
    spray(hHeap, 0x100, TRUE);

    return 0;
}

void spray(HANDLE hHeap, size_t cnt, BOOL trace) {
    void *p;
    for (size_t i = 0; i < cnt; i++) {
        p = HeapAlloc(hHeap, 0x0, SIZE);
        if (trace) {
            printf("HeapAlloc() == %p\n", p);
        }
    }
}
```

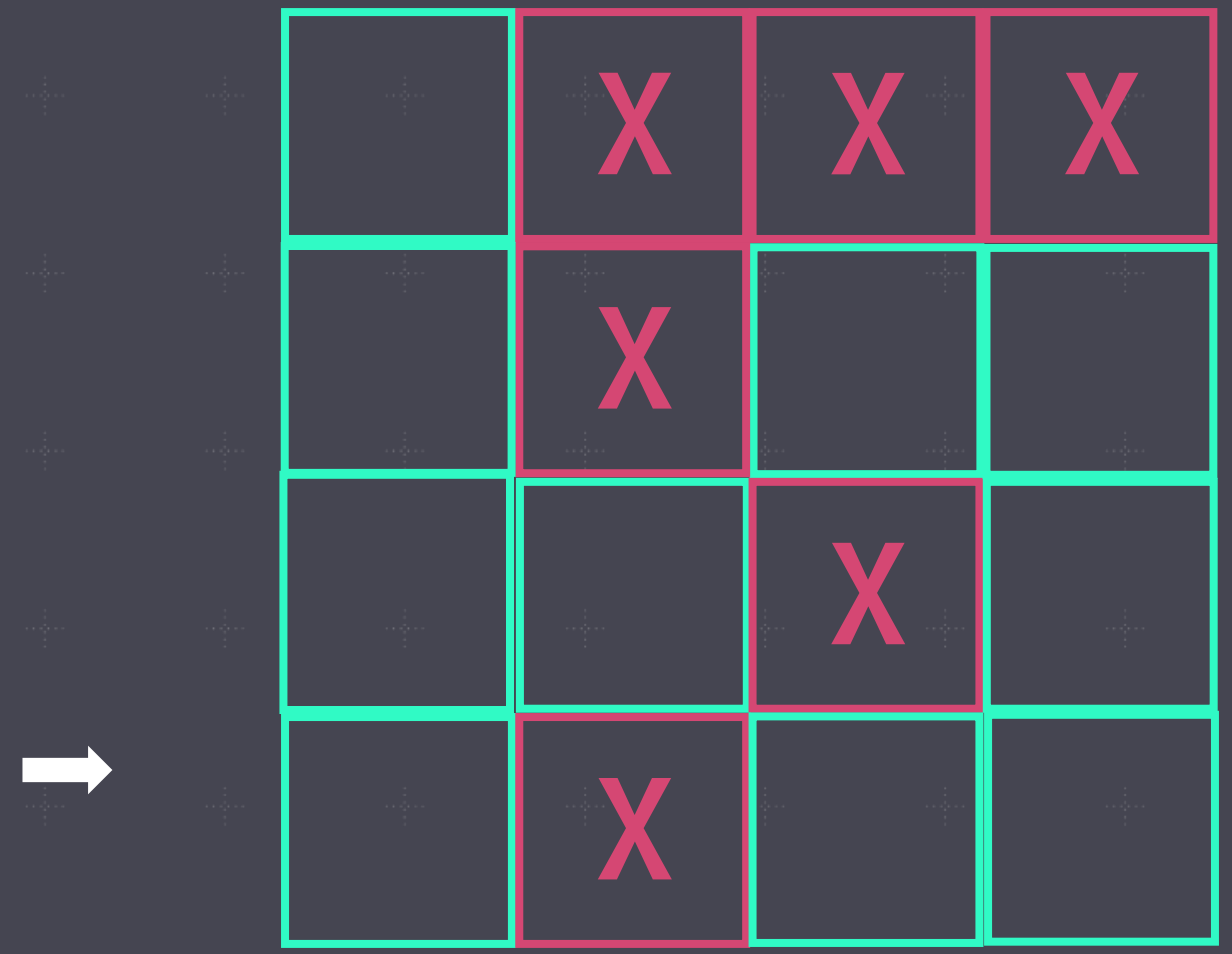
```
[*] activate bucket 0x100 in LFH
[*] spray
HeapAlloc() == 000001540F274B10
HeapAlloc() == 000001540F2747E0
HeapAlloc() == 000001540F2746D0
HeapAlloc() == 000001540F2745C0
HeapAlloc() == 000001540F274C20
HeapAlloc() == 000001540F274A00
HeapAlloc() == 000001540F274D30
HeapAlloc() == 000001540F2748F0
HeapAlloc() == 000001540F273F60
HeapAlloc() == 000001540F274070
HeapAlloc() == 000001540F274180
HeapAlloc() == 000001540F274290
HeapAlloc() == 000001540F2743A0
HeapAlloc() == 000001540F2744B0
HeapAlloc() == 000001540F276CF0
HeapAlloc() == 000001540F275480
HeapAlloc() == 000001540F277130
HeapAlloc() == 000001540F2768B0
HeapAlloc() == 000001540F275BF0
HeapAlloc() == 000001540F275590
HeapAlloc() == 000001540F276470
```

Bypassing LFH randomization on TH1

- How random are these random allocations?
- Seeded from a global random data pool
  - ntdll!RtlpLowFragHeapRandomData
- Pool is CRNG random, but **constant for the lifetime of the process**
- TH1 had an issue that lets us deterministically allocate same or contiguous chunks in memory
- Details && POC: https://github.com/saaramar/Deterministic_LFH

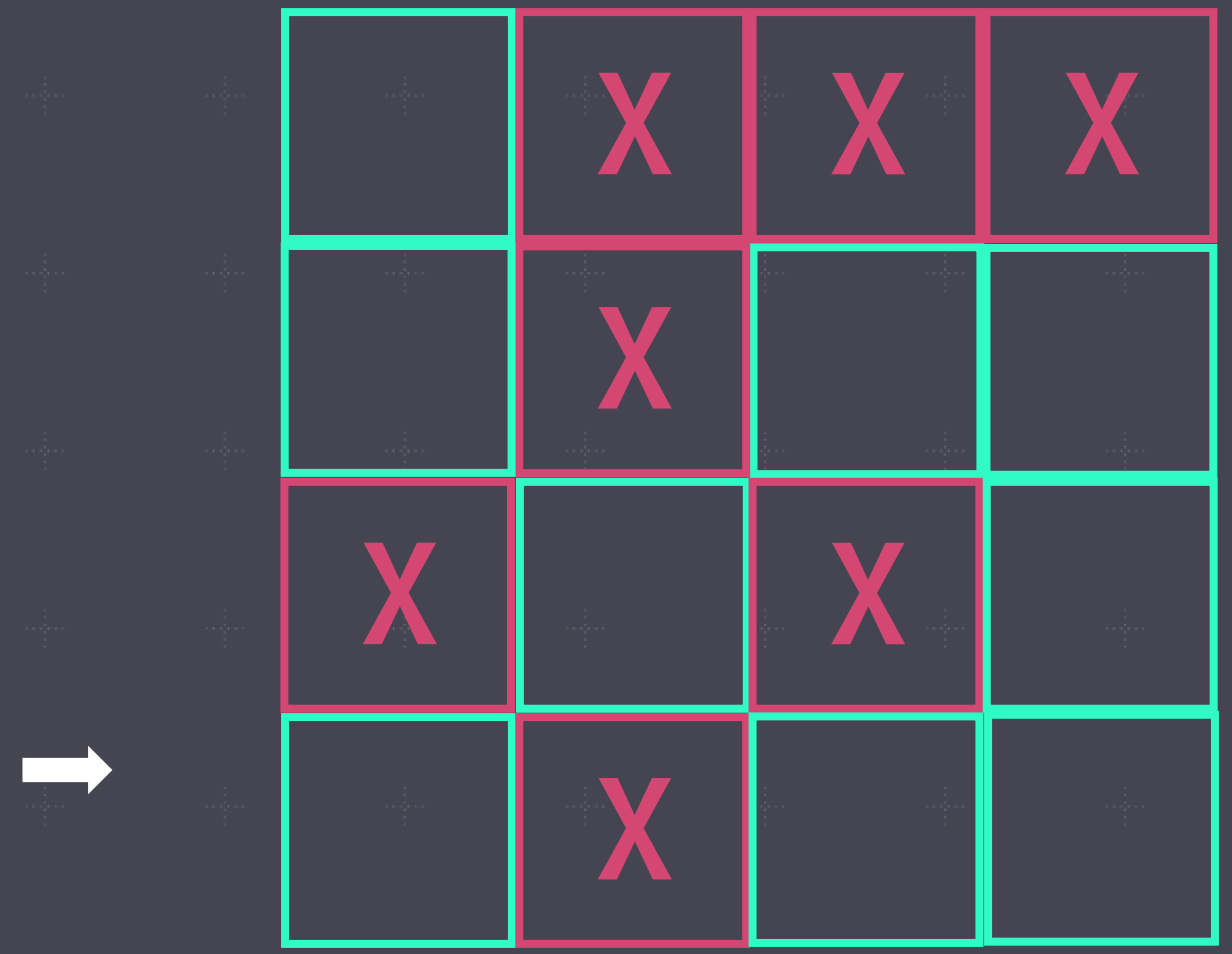
Bypassing LFH randomization on TH1

0	1	1	1
0	1	0	0
0	0	1	0
0	1	0	0

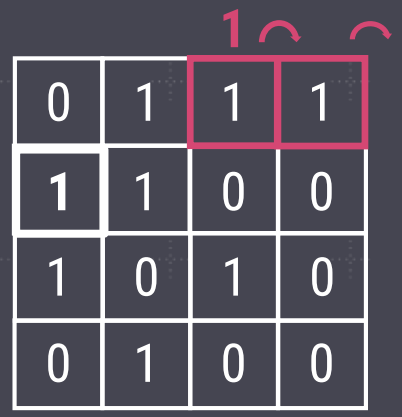


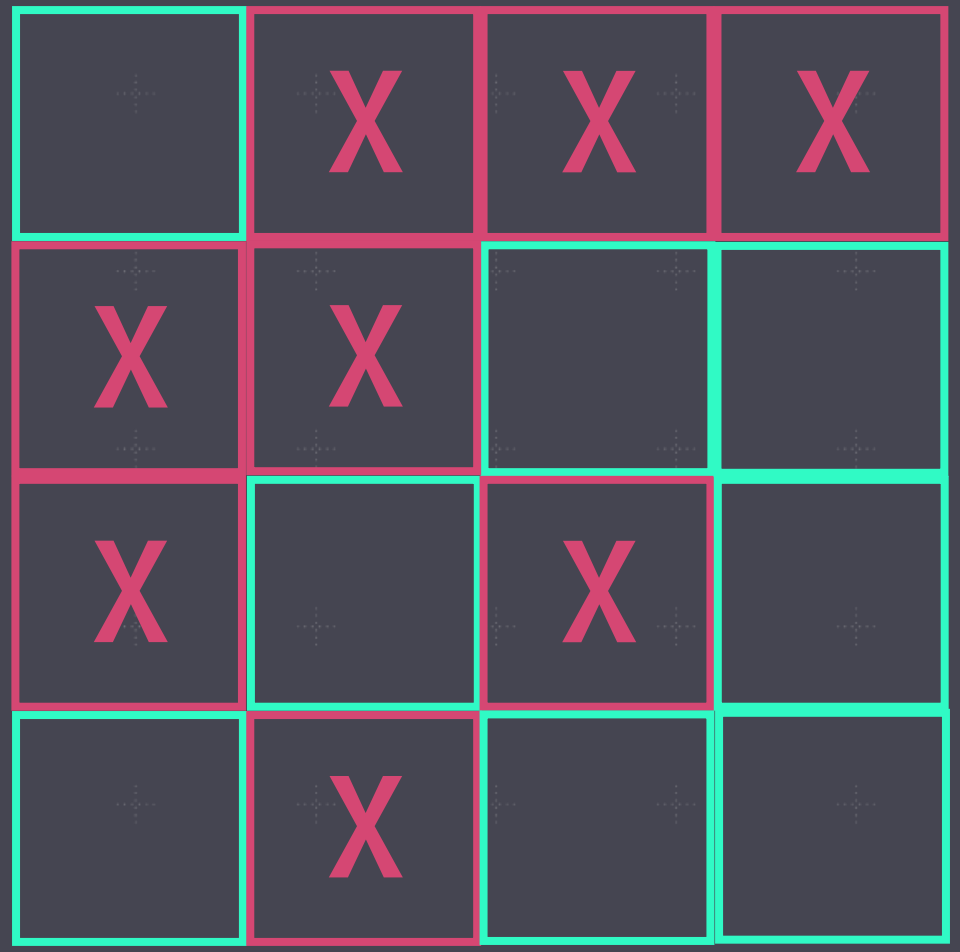
Bypassing LFH randomization on TH1

0	1	1	1
0	1	0	0
1	0	1	0
0	1	0	0

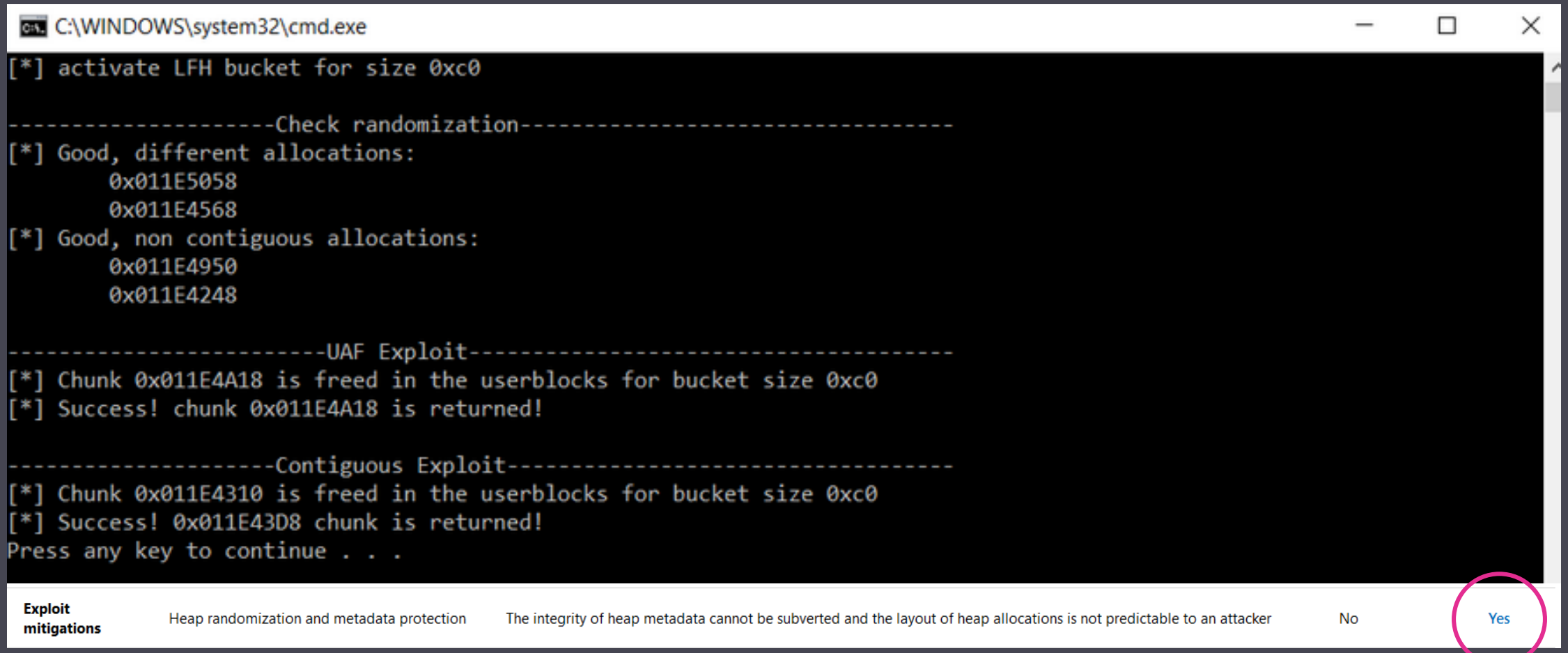


Bypassing LFH randomization on TH1





```
chunk = HeapAlloc(hHeap, 0x0, size);
HeapFree(hHeap, 0x0, chunk);
printf("[*] Chunk 0x%p is freed in the userblocks for bucket size 0x%x\n", chunk, size);

for (size_t i = 0; i < RandomDataArrayLength - 1; ++i) {
    tmp_chunk = HeapAlloc(hHeap, 0x0, size);
    if (!tmp_chunk) {
        return FAIL;
    }
    HeapFree(hHeap, 0x0, tmp_chunk);
}

tmp_chunk = HeapAlloc(hHeap, 0x0, size);
```


C:\WINDOWS\system32\cmd.exe
[*] activate LFH bucket for size 0xc0
----------Check randomization----------
[*] Good, different allocations:
0x011E5058
0x011E4568
[*] Good, non contiguous allocations:
0x011E4950
0x011E4248
----------UAF Exploit----------
[*] Chunk 0x011E4A18 is freed in the userblocks for bucket size 0xc0
[*] Success! chunk 0x011E4A18 is returned!
----------Contiguous Exploit----------
[*] Chunk 0x011E4310 is freed in the userblocks for bucket size 0xc0
[*] Success! 0x011E43D8 chunk is returned!
Press any key to continue . . .
Exploit mitigations
Heap randomization and metadata protection
The integrity of heap metadata cannot be subverted and the layout of heap allocations is not predictable to an attacker
No
Yes

Back to our TH1 exploit

- New LFH shaping strategy for UAF re-allocation
- Should be able to jump to an arbitrary address
  - Using the same function point (onEncounter)
- Let’s jump to a function in ntdll…

```
winworld!Person::encounter+0xeb:
00007ff6`7a0b085b ff155f5c0000    call    qword ptr [winworld!_guard_check_icall_fptr (00007ff6`7a0b64c0)] ds:
0:000> rrcx
rcx=00007fffc9f0552c
0:000> u @rcx
ntdll!RtlLookupFunctionEntry+0x6ec:
00007fff`c9f0552c 654c8b042530000000 mov   r8,qword ptr gs:[30h]
00007fff`c9f05535 4c8d0c24        lea     r9,[rsp]
00007fff`c9f05539 498b4008        mov     rax,qword ptr [r8+8]
00007fff`c9f0553d 488902          mov     qword ptr [rdx],rax
00007fff`c9f05540 498b4010        mov     rax,qword ptr [r8+10h]
00007fff`c9f05544 488901          mov     qword ptr [rcx],rax
00007fff`c9f05547 493bc1          cmp     rax,r9
00007fff`c9f0554a 7708            ja      ntdll!RtlLookupFunctionEntry+0x714 (00007fff`c9f05554)
0:000> p
(e28.d6c): Security check failure or stack buffer overrun - code c0000409 (!!! second chance !!!)
ntdll!_chkstk+0x1a0:
00007fff`c9f86200 cd29            int     29h
```

CFG

NAME

Control Flow Guard

WORKAROUNDS

Known bypasses, see The Evolution of CFI Attacks and Defenses by @JosephBialek

EXPLAINED

Indirect branches are checked against a whitelist of targets, and if the check fails – terminate the process

BOUNTY

Currently out of scope

Windows 10 TH1 Exploit

- Still harder to bypass CFG than to continue with ROP
- Just avoid indirect calls as much as we can
  - Still needed for our *gets* trick, but it is whitelisted
- New exploit:
  - Leak stack address
  - Corrupt return address with the arbitrary write primitive
  - Execute same ROP chain from before
  - PROFIT

WINDOWS 10 TH1 DEMO

WINDOWS 10 RS5

Again…

- Windows 10 TH1 version of the exploit fails on Windows 10 RS5
- The LFH randomization is fixed since build 16179
- However, we already have a strong allocation primitive
- How good will a large random spray be?

```
C:\projects\LFH>LFH_tester.exe
[*] activate bucket 0x100 in LFH
[*] for fun and "fair" game
[*] we passed 20 allocations until we got the last freed chunk
[*] we passed 38 allocations until we got the last freed chunk
[*] we passed 1 allocations until we got the last freed chunk
[*] we passed 4 allocations until we got the last freed chunk
[*] we passed 1 allocations until we got the last freed chunk
[*] we passed 21 allocations until we got the last freed chunk
[*] we passed 27 allocations until we got the last freed chunk
[*] we passed 3 allocations until we got the last freed chunk
[*] we passed 4 allocations until we got the last freed chunk
[*] we passed 8 allocations until we got the last freed chunk
[*] we passed 1 allocations until we got the last freed chunk
[*] we passed 21 allocations until we got the last freed chunk
[*] we passed 30 allocations until we got the last freed chunk
[*] we passed 4 allocations until we got the last freed chunk
[*] we passed 4 allocations until we got the last freed chunk
[*] we passed 3 allocations until we got the last freed chunk
```

Still…

- Execute the new exploit on Windows 10 RS5 it still fails
- This time it’s ACG!

ACG

NAME

Arbitrary Code Guard

WORKAROUNDS

- Known bypasses on older versions
- Execute code in ROP

EXPLAINED

- Restricts allocating and mapping of +X pages
- Restricts editing existing +X pages permissions

BOUNTY

UP TO $45K

CIG

NAME

Code Integrity Guard

WORKAROUNDS

- Use signed DLLs
- Execute code in ROP

EXPLAINED

Restricts loading of unsigned DLLs

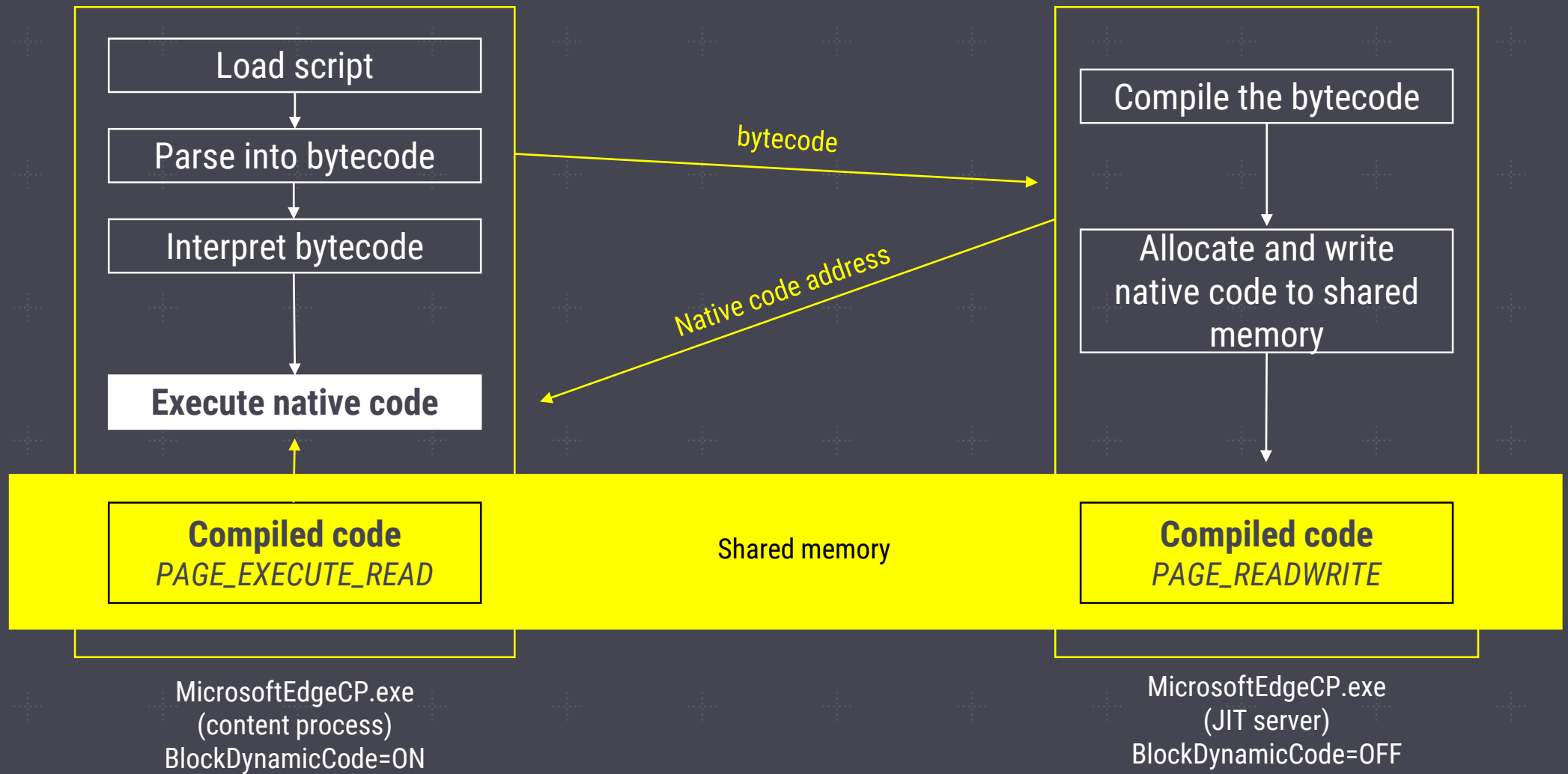
BOUNTY

UP TO $45K

ACG bypasses – Edge use case

- Edge uses separate process for JIT (it has to…)

ACG bypasses – Edge use case



Based on a diagram from "Bypassing Mitigations by Attacking JIT Server in Microsoft Edge" by Ivan Fratric

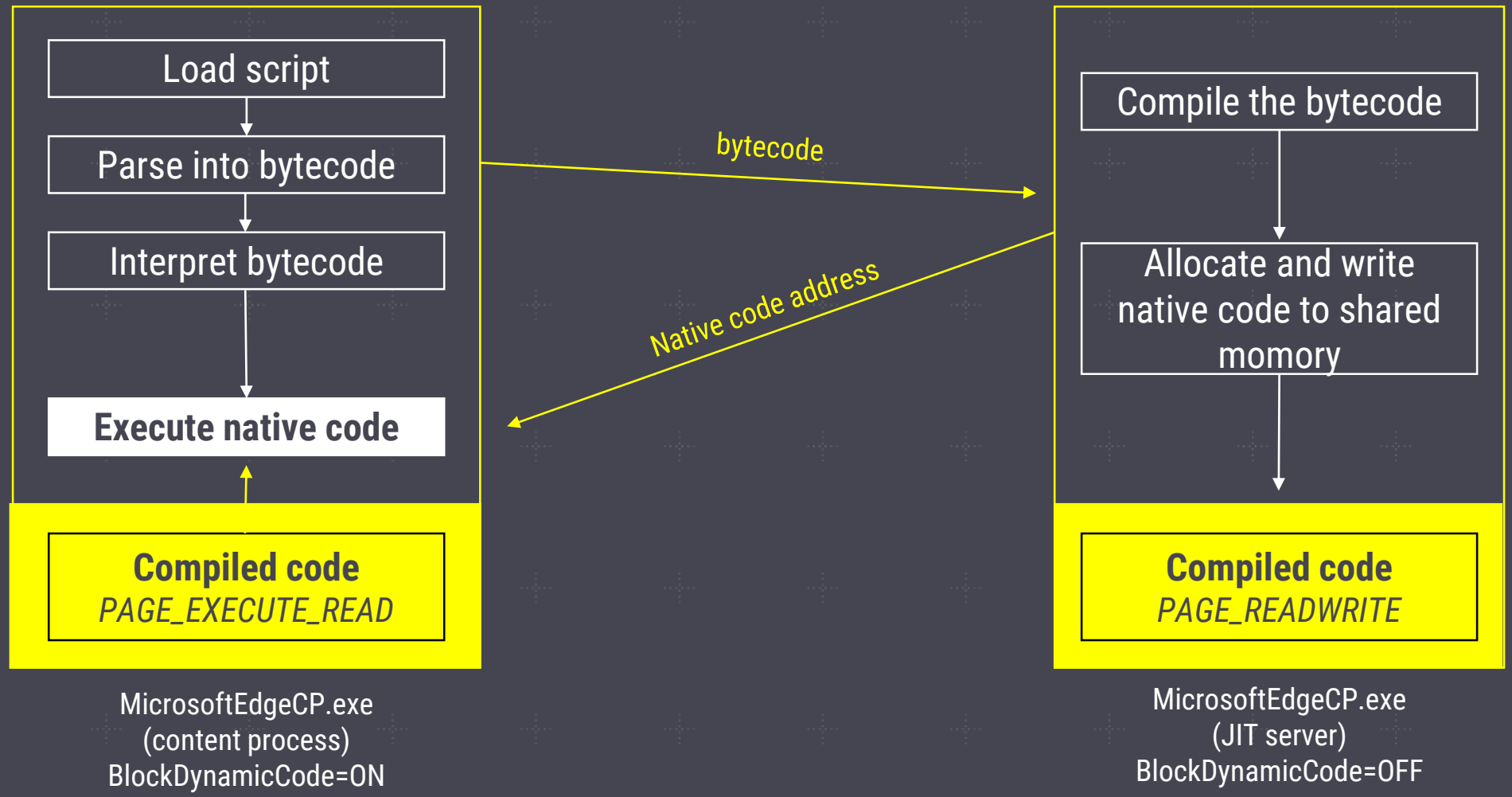
Edge ACG Old bypasses - Duplicate Handle

- Credit: Ivan Fratric, GPZ, issue 1299
- JIT process needs to map +X memory in the calling process
- For that, it must have a handle of the calling process
- In order to send its handle to the JIT process, the calling process first needs to call *DuplicateHandle* on its (pseudo)handle.
- Content process needs to keep the handle of the target process (JIT process) with the PROCESS_DUP_HANDLE access right

Edge ACG Old bypasses - UnmapViewOfFile

- Credit: Ivan Fratric, GPZ, issue 1435
- Unmap the shared memory using *UnmapViewOfFile()*
- Predict the address the JIT server is going to write
- Allocate a writable memory region in the same address
- Write shellcode
- When JIT process calls *VirtualAllocEx()*, even though the memory is already allocated:
  - the call is going to succeed
  - the memory protection is going to be set to PAGE_EXECUTE_READ, content survives!
  - The “true” JITted payload will be written into the JIT server’s "side" of the shared memory

Edge ACG Old bypasses - UnmapViewOfFile



Based on a diagram from "Bypassing Mitigations by Attacking JIT Server in Microsoft Edge" by Ivan Fratric

“Pure” ACG Bypass - Warbird

- Credit: Alex Ionescu
- Added DRM to the kernel a few years ago - Warbird API
  - Accessible from *NtSetSystemInformation(SystemControlFlowInformation)*
- API doesn’t restrict caller
  - can be Low IL, App Container, LPAC…
- Allows creating +X memory inside calling process
- Using MDLs this memory is then made writable
- Caller can load a new trap frame with an arbitrary return address in userspace
  - Nothing checks the CFG bitmap against it.

Working our way around ACG && CIG

- Avoid *VirtualAllocEx / VirtualProtect*
- Executing everything in ROP => no shellcode anymore
- Need a different ROP chain
- Simplest solution would be to execute a process
  - *CreateProcess / ShellExecuteEx / system*
- However…

Child Process Restriction

NAME

Child Process Restriction

WORKAROUNDS

- Implement execve in userspace
- Chain a kernel exploit

EXPLAINED

A child process cannot be created when this restriction is enabled

BOUNTY

UP TO $15K

WINDOWS 10 RS5 DEMO

Even more mitigations

- Containers/sandboxing
  - LPAC, WDAG, etc.
- New improved CFG
- Intel CET to mitigate ROP
  - We tried RFG, had a by-design bypass…
- Many others in kernelspace
- We need your help!

Category	Security feature	Security goal	Intent is to service?	Bounty?
User safety	User Account Control (UAC)	Prevent unwanted system-wide changes (files, registry, etc) without administrator consent	No	No
User safety	AppLocker	Prevent unauthorized applications from executing	No	No
User safety	Controlled Folder Access	Protect access and modification to controlled folders from apps that may be malicious	No	No
User safety	Mark of the Web (MOTW)	Prevent active content download from the web from elevating privileges when viewed locally	No	No
Exploit mitigations	Data Execution Prevention (DEP)	An attacker cannot execute code from non-executable memory such as heaps and stacks	No	Yes
Exploit mitigations	Address Space Layout Randomization (ASLR)	The layout of the process virtual address space is not predictable to an attacker (on 64-bit)	No	Yes
Exploit mitigations	Kernel Address Space Layout Randomization (KASLR)	The layout of the kernel virtual address space is not predictable to an attacker (on 64-bit)	No	No
Exploit mitigations	Arbitrary Code Guard (ACG)	An ACG-enabled process cannot modify code pages or allocate new private code pages	No	Yes
Exploit mitigations	Code Integrity Guard (CIG)	A CIG-enabled process cannot directly load an improperly signed executable image (DLL)	No	Yes
Exploit mitigations	Control Flow Guard (CFG)	CFG protected code can only make indirect calls to valid indirect call targets	No	No
Exploit mitigations	Child Process Restriction	A child process cannot be created when this restriction is enabled	No	Yes
Exploit mitigations	SafeSEH/SEHOP	The integrity of the exception handler chain cannot be subverted	No	Yes
Exploit mitigations	Heap randomization and metadata protection	The integrity of heap metadata cannot be subverted and the layout of heap allocations is not predictable to an attacker	No	Yes
Exploit mitigations	Windows Defender Exploit Guard (WDEG)	Allow apps to enable additional defense-in-depth exploit mitigation features that make it more difficult to exploit vulnerabilities	No	No
Platform lockdown	Protected Process Light (PPL)	Prevent non-administrative non-PPL processes from accessing or tampering with code and data in a PPL process via open process functions	No	No
Platform lockdown	Shielded Virtual Machines	Help protect a VM's secrets and its data against malicious fabric admins or malware running on the host from both runtime and offline attacks	No	No

Contact

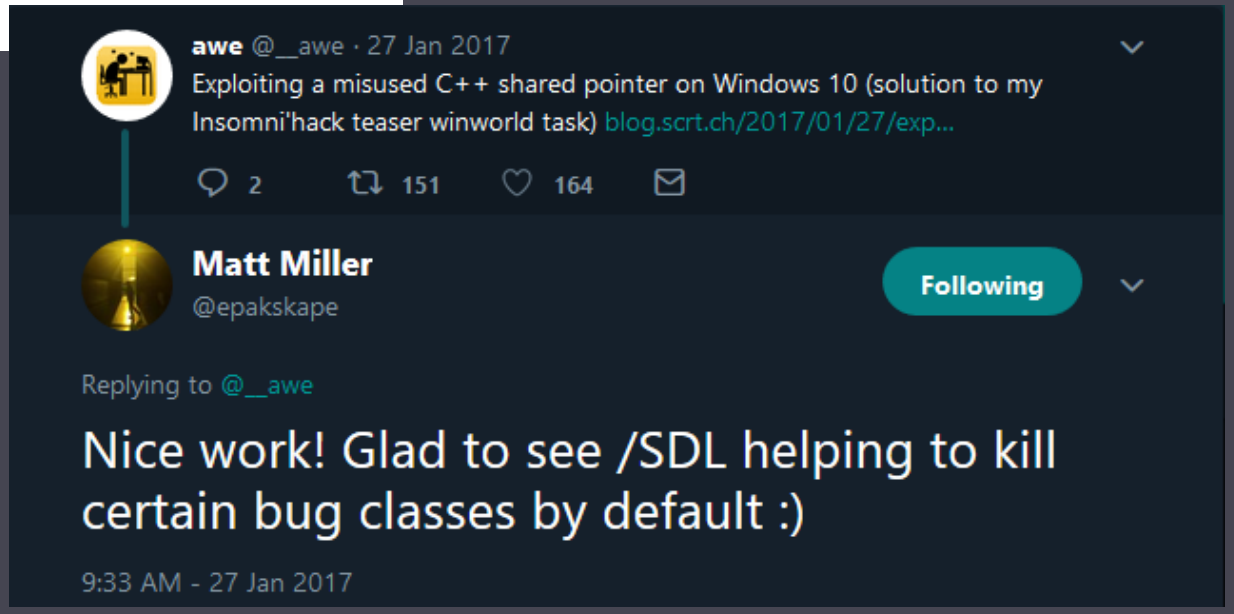
- Send us cool mitigation bypasses to secure@microsoft.com
- Ping me on Twitter, @AmarSaar

- https://www.microsoft.com/en-us/msrc/bounty-mitigation-bypass

Killing bugs before they're born

- From the writeup on the challenge:

> **Note:** It seems that by default Visual Studio 2015 enables the /sdl compilation flag, which will actually add a `memset` to fill the newly allocated `Person` object with zeros, and thus makes it unexploitable. I disabled it 😉 But to be fair, I enabled CFG which isn't default!



Thanks

- @__awe for the awesome challenge
  - We need more Windows CTF challenges!
- @tom41sh for the help and the whiskey

- All of the brilliant engineers who work on mitigations
  - And keeps our lives interesting as exploit developers ☺

Refs

- Security Servicing Criteria for Windows / Bounty Mitigation Bypass
- JIT Server / JIT Server whitepaper
- The Evolution of CFI Attacks and Defenses
- The “Bird” that killed ACG
- Deterministic LFH
- If you like Hebrew (Sorry!) – LFH internals and exploitation
- Exploiting a misused C++ shared pointer on Windows 10

THANKS!