



Bypassing Mitigations by Attacking JIT Server in Microsoft Edge

Ivan Fratric

Infiltrate 2018

About me

- Security researcher at Google Project Zero
- Previously: Google Security Team, Academia (UNIZG)
- Doing security research for the last 10 years
- Author: Domato, WinAFL, ROPGuard
- @ifsecure on Twitter

Browser exploit flow (example)



Browser exploit flow (example)



Code execution mitigations



- Before:

```
mov rbx,qword ptr [rax+8]
call rbx
```

- After:

```
mov rbx,qword ptr [rax+8]
mov rax, rbx
call qword ptr [chakra!_guard_dispatch_icall_fptr]
```

- Bitmap of CFG-allowed targets (some granularity involved)
- Only checks forward edges (doesn't check return addresses)

Code execution mitigations



- 2 new mitigations Introduced in Windows 10 creators update (1703)

Code execution mitigations



- Arbitrary Code Guard (ACG)
- Make it impossible to
 - allocate new executable pages
 - e.g. `VirtualAlloc(... , ... , PAGE_EXECUTE_READWRITE , ...)`
 - make existing executable pages writable
 - e.g. `VirtualProtect(... , ... , PAGE_EXECUTE_READWRITE , ...)`
- Attempting results in `0xc0000604 STATUS_DYNAMIC_CODE_BLOCKED`
- Similar to PaX MPROTECT
- What about JIT? JIT Server.

Code execution mitigations



- Code Integrity Guard (CIG)
 - Can only load properly signed DLLs

Agenda

- How ACG works?
- Is it effective?
- How does JIT server work
- Issues (CFG and ACG)

Enabling ACG

- Relies on setting the dynamic code policy
- Enabled by `SetProcessMitigationPolicy()`
- In Edge:

```
00 KERNELBASE!SetProcessMitigationPolicy
01 MicrosoftEdgeCP!SetProcessDynamicCodePolicy+0xc0
02 MicrosoftEdgeCP!StartContentProcess_Exe+0x164
03 MicrosoftEdgeCP!main+0xfe
04 MicrosoftEdgeCP!_main+0xa6
05 MicrosoftEdgeCP!WinMainCRTStartup+0x1b3
06 KERNEL32!BaseThreadInitThunk+0x14
07 ntdll!RtlUserThreadStart+0x21
```

When is it enabled?

```
Windows PowerShell
PS C:\Users\Ivan Fratric> Get-ProcessMitigation 2416

ProcessName      : MicrosoftEdgeCP
Source           : Running Process
Id              : 2416

DEP:
  Enable          : ON
  EmulateAtlThunks : ON
  Override DEP    : False

ASLR:
  BottomUp        : ON
  Override BottomUp : False
  ForceRelocateImages : NOTSET
  RequireInfo     : NOTSET
  Override ForceRelocate : False
  HighEntropy     : ON
  Override High Entropy : False

StrictHandle:
  Enable          : ON
  Override StrictHandle : False

System Call:
  DisableWin32kSystemCalls : OFF
  Audit                   : OFF
  Override SystemCall      : False

ExtensionPoint:
  DisableExtensionPoints : OFF
  Override ExtensionPoint : False

DynamicCode:
  BlockDynamicCode      : ON
  AllowThreadsToOptOut  : OFF
  Audit                 : OFF
  Override DynamicCode  : False

CFG:
  Enable          : ON
  SuppressExports : ON
  Override CFG    : False
  StrictControlFlowGuard : OFF
  Override StrictCFG : False
```

DynamicCode:

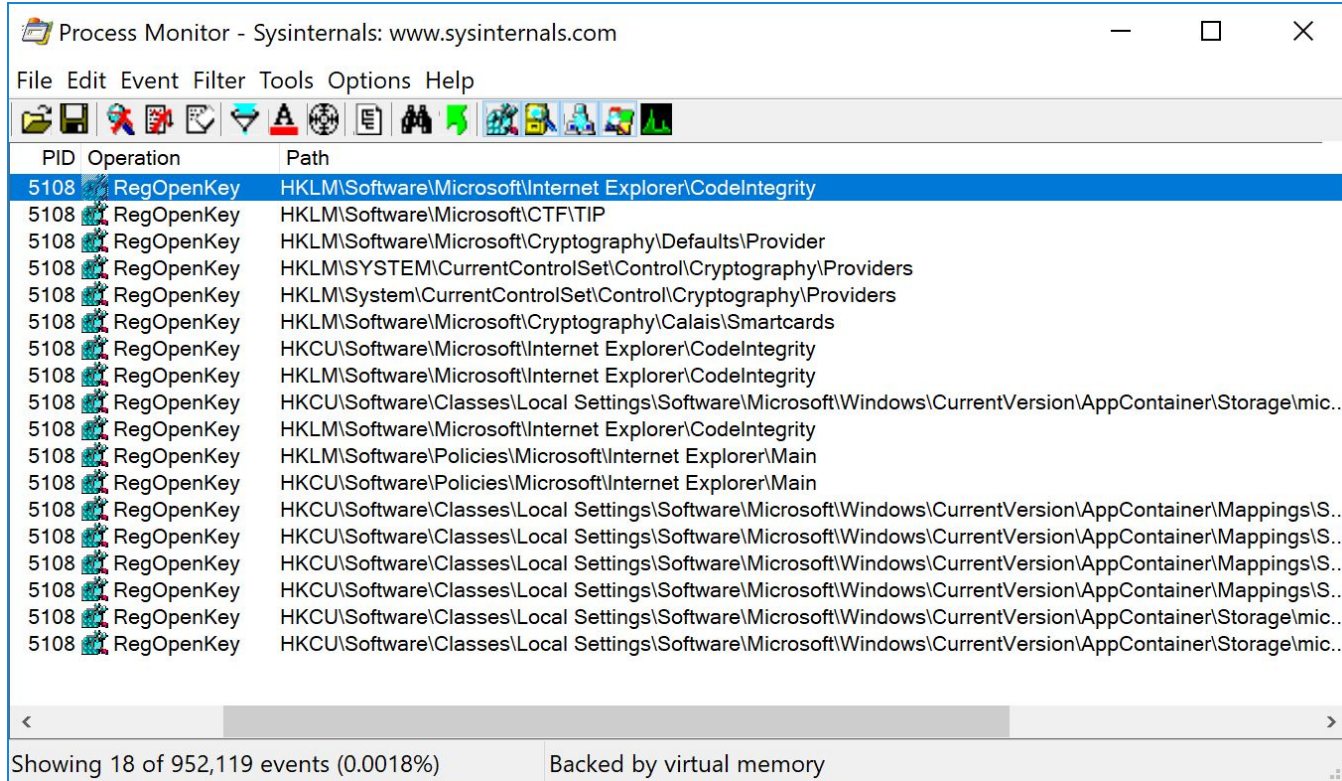
BlockDynamicCode	: ON
AllowThreadsToOptOut	: OFF
Audit	: OFF
Override DynamicCode	: False

When is it enabled?

From Microsoft's blog post:

For compatibility reasons, ACG is currently only enforced on 64-bit desktop devices with a primary GPU running a WDDM 2.2 driver (the driver model released with the Windows 10 Anniversary Update), or when software rendering is use. For experimental purposes, software rendering can be forced via Control Panel -> Internet Options -> "Advanced". Current Microsoft devices (Surface Book, Surface Pro 4, and Surface Studio) as well as a few other existing desktop systems with GPU drivers known to be compatible with ACG are opted into ACG enforcement. We intend to improve the coverage and accuracy of the ACG GPU opt-in list as we evaluate the telemetry and feedback from customers.

When is it enabled?



The screenshot shows the Process Monitor application window with the title bar "Process Monitor - Sysinternals: www.sysinternals.com". The menu bar includes "File", "Edit", "Event", "Filter", "Tools", "Options", and "Help". The toolbar contains various icons for file operations, filters, and settings. The main display area is a table with three columns: "PID", "Operation", and "Path". The table lists 18 events, all with PID 5108 and Operation "RegOpenKey". The paths are various registry locations, including HKLM\Software\Microsoft\Internet Explorer\CodeIntegrity, HKLM\Software\Microsoft\CTF\TIP, HKLM\Software\Microsoft\Cryptography\Defaults\Provider, HKLM\SYSTEM\CurrentControlSet\Control\Cryptography\Providers, HKLM\System\CurrentControlSet\Control\Cryptography\Providers, HKLM\Software\Microsoft\Cryptography\Calais\Smartcards, HKCU\Software\Microsoft\Internet Explorer\CodeIntegrity, HKLM\Software\Microsoft\Internet Explorer\CodeIntegrity, HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Storage\mic..., HKLM\Software\Microsoft\Internet Explorer\CodeIntegrity, HKLM\Software\Policies\Microsoft\Internet Explorer\Main, HKCU\Software\Policies\Microsoft\Internet Explorer\Main, HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Mappings\S..., HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Mappings\S..., HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Mappings\S..., HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Mappings\S..., HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Storage\mic..., and HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Storage\mic... The status bar at the bottom shows "Showing 18 of 952,119 events (0.0018%)" and "Backed by virtual memory".

PID	Operation	Path
5108	RegOpenKey	HKLM\Software\Microsoft\Internet Explorer\CodeIntegrity
5108	RegOpenKey	HKLM\Software\Microsoft\CTF\TIP
5108	RegOpenKey	HKLM\Software\Microsoft\Cryptography\Defaults\Provider
5108	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Control\Cryptography\Providers
5108	RegOpenKey	HKLM\System\CurrentControlSet\Control\Cryptography\Providers
5108	RegOpenKey	HKLM\Software\Microsoft\Cryptography\Calais\Smartcards
5108	RegOpenKey	HKCU\Software\Microsoft\Internet Explorer\CodeIntegrity
5108	RegOpenKey	HKLM\Software\Microsoft\Internet Explorer\CodeIntegrity
5108	RegOpenKey	HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Storage\mic...
5108	RegOpenKey	HKLM\Software\Microsoft\Internet Explorer\CodeIntegrity
5108	RegOpenKey	HKLM\Software\Policies\Microsoft\Internet Explorer\Main
5108	RegOpenKey	HKCU\Software\Policies\Microsoft\Internet Explorer\Main
5108	RegOpenKey	HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Mappings\S...
5108	RegOpenKey	HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Mappings\S...
5108	RegOpenKey	HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Mappings\S...
5108	RegOpenKey	HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Mappings\S...
5108	RegOpenKey	HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Storage\mic...
5108	RegOpenKey	HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppContainer\Storage\mic...

Showing 18 of 952,119 events (0.0018%) Backed by virtual memory

How effective is ACG?

Assumption: Attacker has a read/write primitive

- Data-only attacks
- Code reuse attacks
 - Do we need a ROP compiler?
- Code second-stage payloads in JavaScript
 - Need a way to call native-code functions from JavaScript and continue running script
 - Libraries already exist (pwn.js from Theori)

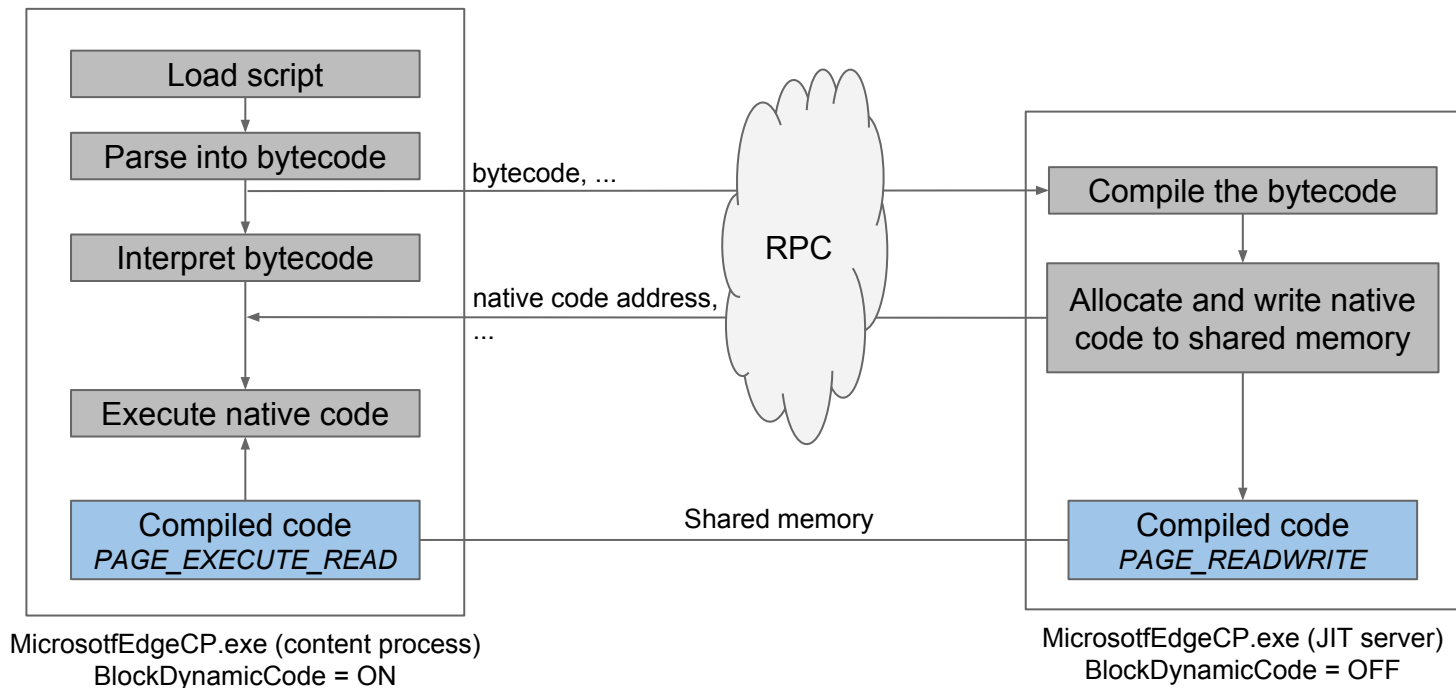
Mitigations that work together

- ACG, CIG, no CFG => ROP, privescs in JavaScript
- CFG, CIG, no ACG => Overwrite/allocate executable memory
- CFG, ACG, no CIG => Load a malicious .dll

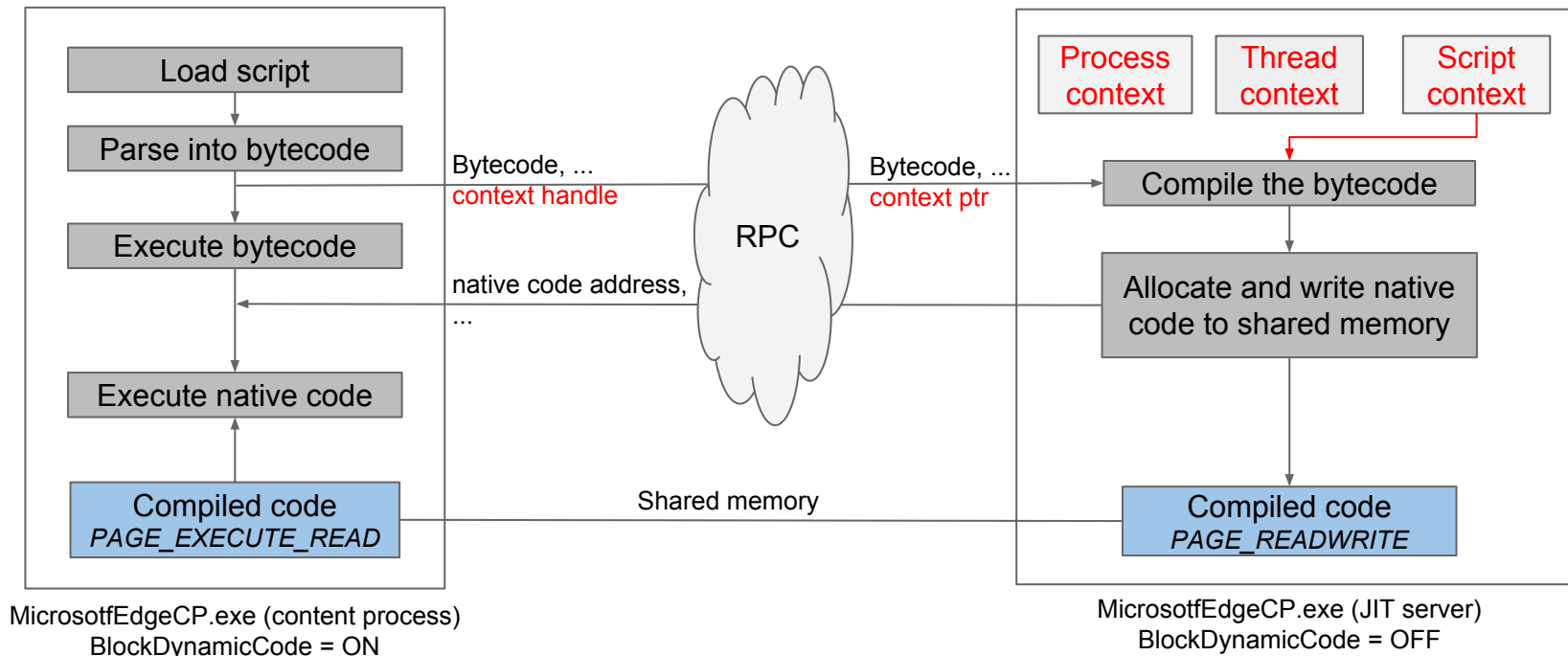
ACG Bypasses, prior work

- Abusing thread opt-out (no longer the case)
- Bypass using Warbird DRM framework (Alex Ionescu)

JIT server (simplified)



JIT server, maintaining state



Exposed methods / managing contexts

- (!) ConnectProcess - Connects a new Content Process and creates the corresponding Process Context
- (!) InitializeThreadContext - Creates a ServerThreadContext object on the server. Also pre-reserves memory for compiled code and JIT thunks.
- InitializeScriptContext - Creates a ServerThreadContext object on the server.
- CleanupThreadContext - Marks Thread context as closed, removes it from the Thread context dictionary and closes all associated ScriptContexts
- CloseScriptContext - Marks Script context as closed and removes it from the Script context dictionary
- CleanupScriptContext - Closes script context if not closed already and deletes the associated ServerScriptContext object
- Shutdown - Deletes closed context objects, deletes allocated pages and unregisters RPC server

Exposed methods / updating data in contexts

- UpdatePropertyRecordMap
- AddDOMFastPathHelper
- AddModuleRecordInfo
- SetWellKnownHostTypeId
- SetIsPRNGSeeded

Exposed methods / working with thunks

- Thunk == short trampoline that jumps to function implementation
 - Executable code
 - Every function gets one
- NewInterpreterThunkBlock - Allocates a new executable buffer and fills it with interpreter thunks.
- DecommitInterpreterBufferManager - Decommits all memory pages used for thunk allocations.
- IsInterpreterThunkAddr - Checks if address is in one of the interpreter thunk blocks

Exposed methods / working with compiled code

- (!) RemoteCodeGen
 - This is where the magic happens
 - Large structure as input/output
 - Bytecode
 - Type information, caches, inlinee information, addresses
- IsNativeAddr - checks if address is in one of the JIT blocks
- (!) FreeAllocation - Frees executable memory allocation made previously by the server and clears CFG targets

JIT phases (1/2)

- (!) Build Intermediate Representation (IR) from bytecode
- Function inlining
- Build flow graph
- Global optimizations
- Lower IR into machine-specific representation (not yet encoded)
- Encode large constants (security)
- Insert stack probes
- Register allocation

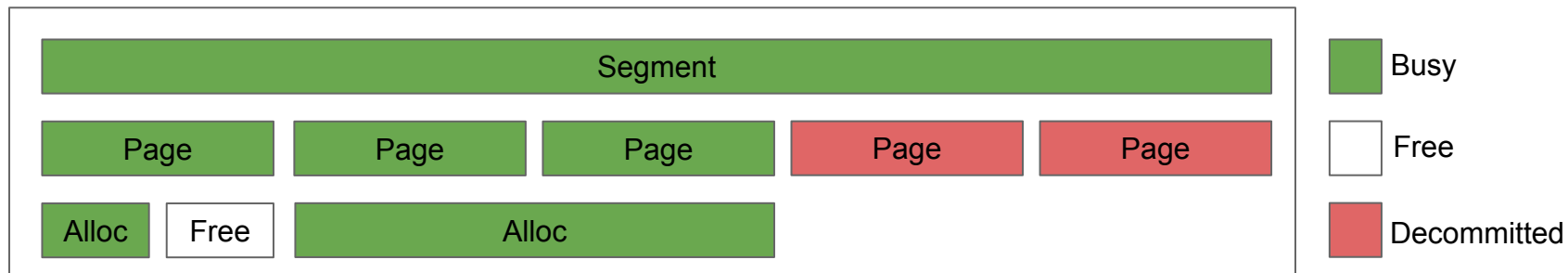
JIT phases (2/2)

- Peephole optimizations
- Layout
- Insert bailouts
- Insert NOPs at random points (security)
- Insert function prolog and epilog
- Final lower
- (!) Encoder
- Fixups on data allocated by JIT process

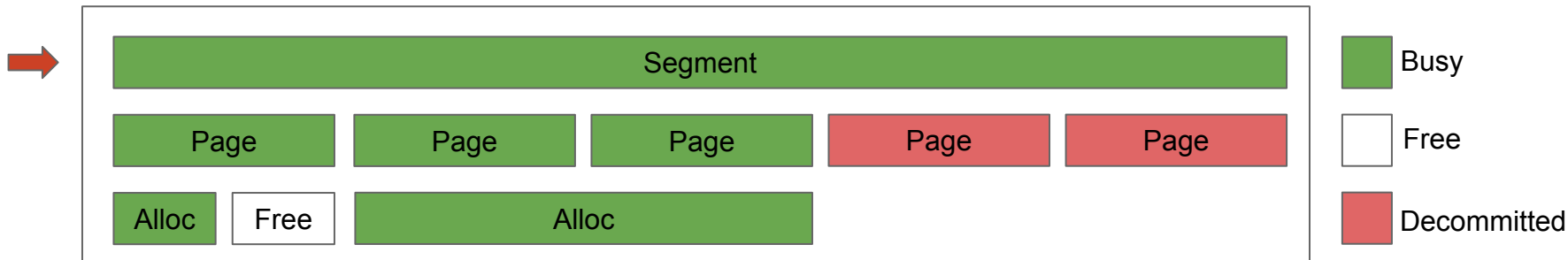
Encoder phase (Encoder.cpp)

- Prepares the buffer with compiled code
 - Encoded instructions
 - Jump tables for switch statements
- Allocates memory for executable code
- Copies the buffer

Allocating memory

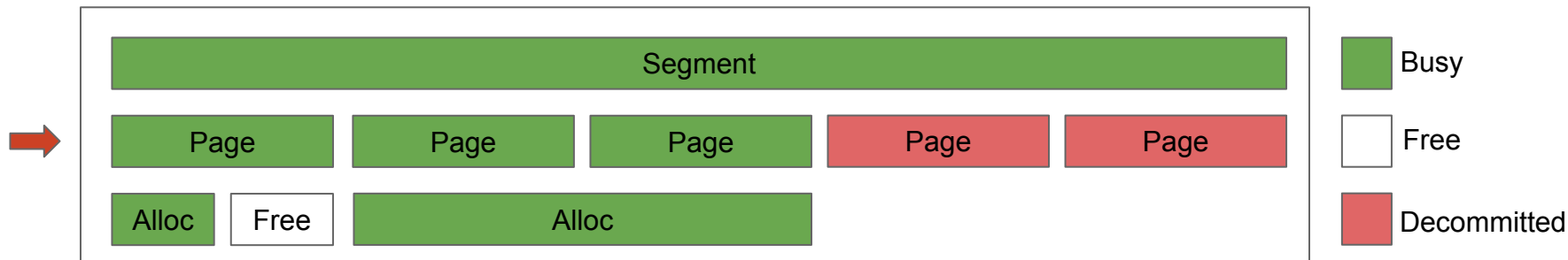


Allocating memory / Segments (SectionAllocWrapper.cpp)



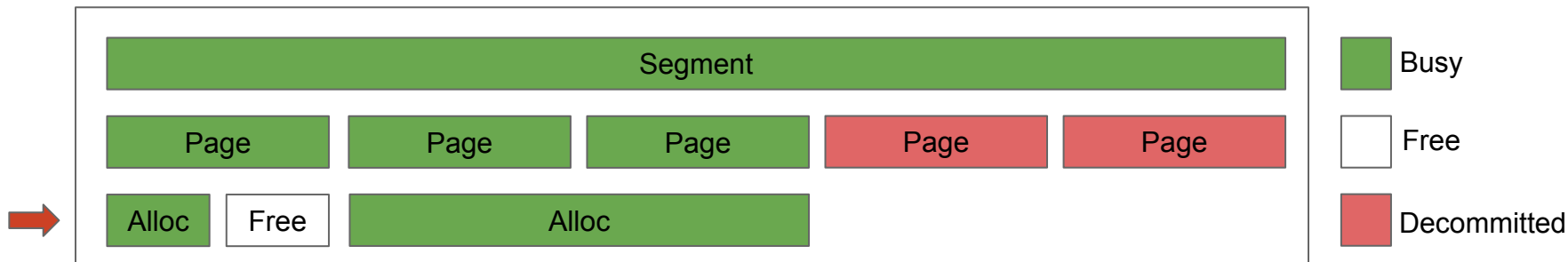
- Segment == Shared memory object (created via CreateFileMapping)
- Mapped into each process using MapViewOfFile2
 - PAGE_EXECUTE_READ for content process
 - PAGE_READWRITE for JIT process
- In JIT process unmapped immediately after writing

Allocating memory / Pages (PageAllocator.cpp)



- Pages start as decommitted -> committed using VirtualAllocEx when needed
- Each segment has 2 bit vectors for free pages and decommitted pages
- Once a page gets committed it gets filled with 0xCC (int 3)
- When sufficient number of pages is freed, pages start getting decommitted

Allocating memory / Allocations (CustomHeap.cpp)



- Large allocations ($>\text{pagesize}$) get the corresponding number of pages
- For smaller allocations, pages get divided into 128-byte blocks
 - Bitmap of free blocks inside a page
- Pages get put in buckets for allocations of size 128, 256, 512, 1024, 2048, 4096
- Metadata is not stored together with data, stored in Allocation objects on the server only
- Upon freeing, data is filled with 0xCC (int 3)

Issues

- CFG
 - Issues that rely on return address overwrite
 - Issues that don't rely on return address overwrite
- ACG
 - Memory corruption issues in the JIT process
 - Logic issues in the JIT process

Controlling bytecode

- What can we do with bytecode?
- T. Dullien: “Exploitation and state machines”
 - Arbitrary read/write
 - Overwriting the stack (in Chakra e.g. OP_ArgOut_A)

Call instructions in the JIT code

- What happens when JIT code needs to call a function, e.g.

```
call chakra!helper_function
```

- JIT server needs to know address of DLLs in the Content Process
 - Q: How does it know?
 - A: Content Process tells it.

Checking module address in Content Process

- VirtualQueryEx on the first page and check:
 - Return value of VirtualQueryEx is correct
 - allocation base address is the same as provided by the client
 - memory type is MEM_IMAGE
 - memory state is MEM_COMMIT
 - region size is not smaller than 4096 bytes
- Get image headers and check:
 - number of sections is correct
 - number of symbols is correct
 - checksum in the header is correct
 - image size is correct
- Bypassable by modifying the header region of another module

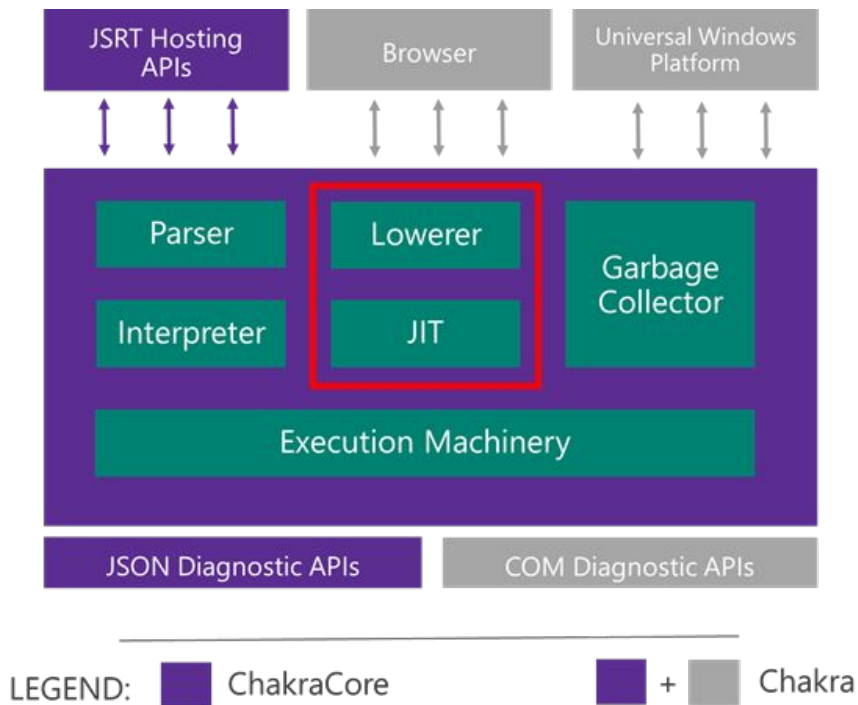
Dangling CFG target

- From ServerFreeAllocation:

```
context->SetValidCallTargetForCFG((PVOID)codeAddress, false);  
context->GetCodeGenAllocators()->emitBufferManager.FreeAllocation((void*)codeAddress);
```

- codeAddress inside allocation -> FreeAllocation() succeeds
 - But CFG target doesn't get unset
- Possible to free allocation without clearing CFG flags

JIT server attack surface



Memory corruption issues

- Integer overflows (CVE-2017-8637)

```
offsetToInstructionCount = lastOffset + 2;  
m_offsetToInstruction = JitAnewArrayZ(m_tempAlloc, IR::Instr *, offsetToInstructionCount);  
  
m_saveLoopImplicitCallFlags = (IR::Opnd**)func->m_alloc->Alloc(sizeof(IR::Opnd*) * loopCount);  
this->tempMap = (SymID*)m_tempAlloc->AllocZero(sizeof(SymID) * tempCount);
```

- Out-of-bound writes (CVE-2017-8659)

```
this->m_saveLoopImplicitCallFlags[num] = saveOpnd;
```

- Bytecode fuzzing produces crashes

Memory corruption issues

- Does it make sense to exploit another memory corruption bug?
- Pros:
 - Lots of them
 - ASLR already bypassed
- Cons:
 - CFG
 - Heap ASLR
 - Exploit stability

The trouble with handles

- JIT Server needs to be able to allocate memory in Content Process
 - JIT Server has a handle to Content Process
- Content Process needs to give its handle
 - Needs to call DuplicateHandle() first
- Content Process needs a handle to JIT server to call DuplicateHandle()
 - ...with PROCESS_DUP_HANDLE permissions

The trouble with handles

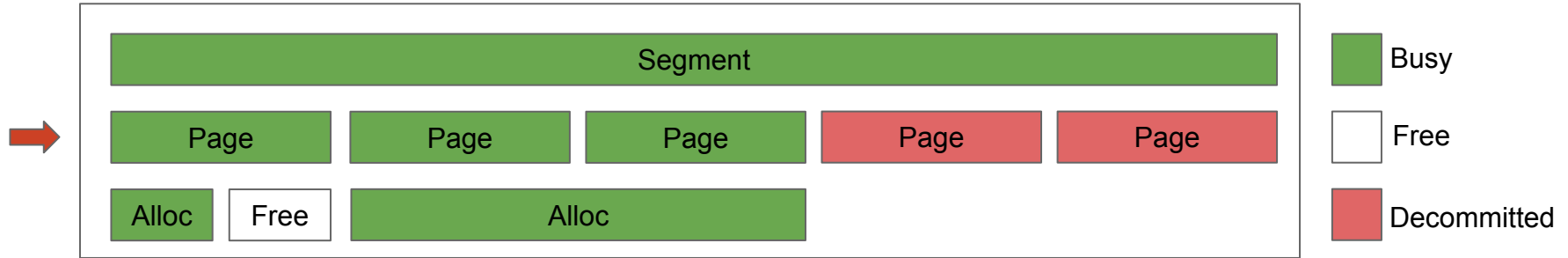


The trouble with handles

- The issue:

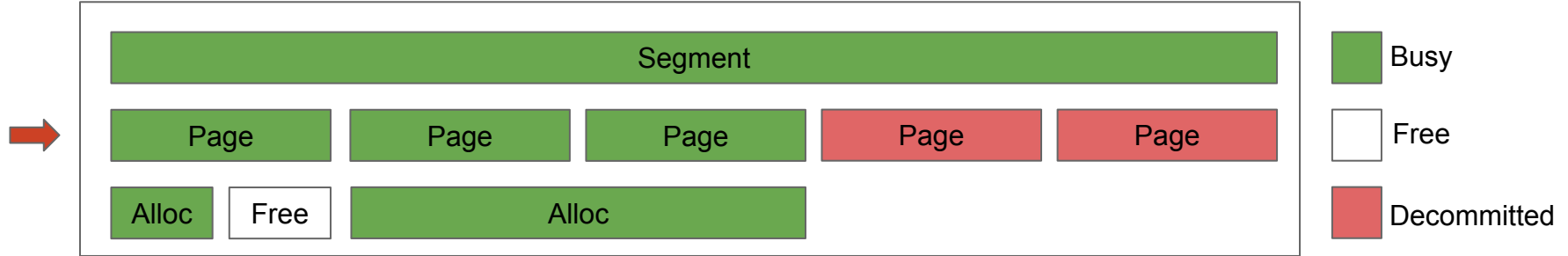
Warning A process that has some of the access rights noted here can use them to gain other access rights. For example, if process A has a handle to process B with **PROCESS_DUP_HANDLE** access, it can duplicate the pseudo handle for process B. This creates a handle that has maximum access to process B. For more information on pseudo handles, see [GetCurrentProcess](#).

Exploiting memory management



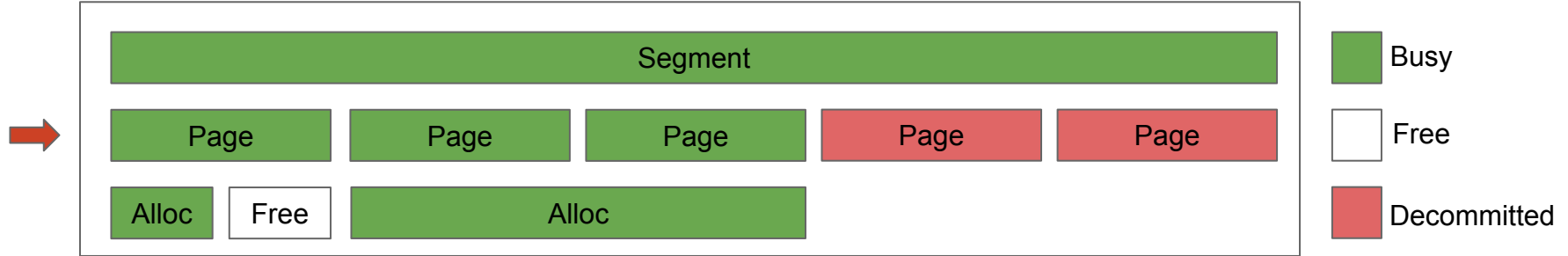
- Pages start as decommitted -> committed using VirtualAllocEx when needed
- Each segment has 2 bit vectors for free pages and decommitted pages
- Once a page gets committed it gets filled with 0xCC (int 3)
- When sufficient number of pages is freed, pages start getting decommitted

Exploiting memory management



- Pages start as decommitted -> committed using **VirtualAllocEx** when needed
- Each segment has 2 bit vectors for free pages and decommitted pages
- Once a page gets committed it gets filled with 0xCC (int 3)
- When sufficient number of pages is freed, pages start getting decommitted

Exploiting memory management



- Pages start as decommitted -> committed using **VirtualAllocEx** when needed
 - VirtualAllocEx called with flProtect = **PAGE_EXECUTE_READ**

Exploiting memory management

- Predict the address of next JIT allocation.
- Unmaps the shared memory with `UnmapViewOfFile()`
- Allocate same pages with `PAGE_READWRITE`
- Write payload
- Wait

Pid 812 - WinDbg:10.0.14321.1024 AMD64

File Edit View Debug Window Help

Command

```

Mapping TEB and stack regions...
Mapping heap regions...
*** Failure in mapping Heap (80004002: ExtRemoteTyped::
Mapping page heap regions...
Mapping other regions...
Mapping stack trace database regions...
Mapping activation context regions...

Usage:                <unknown>
Base Address:         00000225`80007000
End Address:          00000225`80009000
Region Size:          00000000`00002000 ( 8.000 kB)
State:                00001000          MEM_COMMIT
Protect:              00000010          PAGE_EXECUTE
Type:                 00020000          MEM_PRIVATE
Allocation Base:      00000225`80000000
Allocation Protect:   00000004          PAGE_READWRITE

Content source: 1 (target), length: ff6
0:032>

```

Memory

Virtual:	@\$scopeip	Previous	Display format:	Byte	Next
00000225`8000700a	fe 00 41 43 47 20			..ACG	
00000225`80007010	42 59 50 41 53 53			BYPASS	
00000225`80007016	20 48 b8 37 13 37			H.7.7	
00000225`8000701c	13 37 13 00 00 fe			.7....	
00000225`80007022	00 41 43 47 20 42			.ACG B	
00000225`80007028	59 50 41 53 53 20			YPASS	
00000225`8000702e	48 b8 37 13 37 13			H.7.7.	
00000225`80007034	37 13 00 00 fe 00			7....	
00000225`8000703a	41 43 47 20 42 59			ACG BY	
00000225`80007040	50 41 53 53 20 48			PASS H	

Disassembly

Offset: @\$scopeip

```

add     byte ptr [rax],al
add     byte ptr [rax],al
add     byte ptr [rax],al
add     byte ptr [rax],al
130000 mov     rax,133713371337h
inc     byte ptr [rax] ds:00001337`13371337
???
???
and     byte ptr [r10+59h],r8b

```

Ln 0, Col 0 Sys 0:<Local> Proc 000:32c Thrd 032:a08 ASM OVR CAPS NUM

Enabling ACG

- Relies on setting the dynamic code policy
- Enabled by `SetProcessMitigationPolicy()`
- In Edge:

```
00 KERNELBASE!SetProcessMitigationPolicy
01 MicrosoftEdgeCP!SetProcessDynamicCodePolicy+0xc0
02 MicrosoftEdgeCP!StartContentProcess_Exe+0x164
03 MicrosoftEdgeCP!main+0xfe
04 MicrosoftEdgeCP!_main+0xa6
05 MicrosoftEdgeCP!WinMainCRTStartup+0x1b3
06 KERNEL32!BaseThreadInitThunk+0x14
07 ntdll!RtlUserThreadStart+0x21
```

Disabling ACG

- ACG gets enabled too early for the Content Process to disable it for itself
- But...
 - James Forshaw: Did you know one Content Process can open another?
 - Me: Nah, I tried that, didn't work
 - [try again]
 - Me: Oh, snap...
- One MicrosoftEdgeCP.exe can disable ACG in another MicrosoftEdgeCP.exe
 - Both processes need to be in the same App Container
 - True for Internet sites
 - The race is easily winnable

Conclusion

- ACG needs strong CFG to be effective
- Attacker's perspective: Business as usual (mostly)
 - Abundant CFG bypasses + calling native functions with JavaScript
 - Implementation issues, large attack surface of the JIT server
- What can Microsoft do
 - Make CFG useful (RFG? CET?)
 - Stronger Content Process <-> JIT Process boundary