AVENGERCON II

Exploit Mitigations in Windows 10 Creators Update

Christian Sharpsten

November 15, 2017

Exploit Mitigations in Windows 10 Creators Update
Christian Sharpsten
November 15, 2017

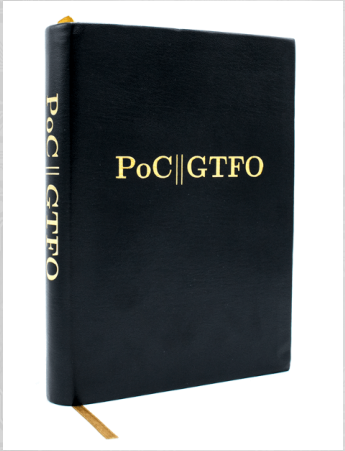
Table of Contents

Table of Contents

1 Exploitation Basics and Mitigations Overview
2 CIG/ACG Implementation Details
3 ACG from Usermode
4 ACG from Kernelmode

- Exploitation basics and modularity
- Brief mitigations overview
- How to enable CIG/ACG?
- Where is CIG/ACG used?
- How is CIG/ACG implemented under the hood?
- How might we get around it?

POC || GTFO



https://github.com/shareef12

POC || GTFO

https://github.com/shareef12

- All code associated with this presentation will be publicly available on github.

Exploit Primitives

- Some kind of write or memory corruption
- Arbitrary write
- Info leak or arbitrary read?
- ROP/limited code execution
- Arbitrary code execution

Exploit Primitives
- Some kind of write or memory corruption
- Arbitrary write
- Info leak or arbitrary read?
- ROP/limited code execution
- Arbitrary code execution

- Once you have usermode execution, you can chain that with privesc to kernel
- Once in kernel, you can persist yourself, or migrate to a better process to live in
- Writing software is a game of interfaces. We want to define our interfaces that allow maximum interoperability and code reuse.

Pre Windows 10

- EMET - DEP
- EMET - Structured Exception Handling Overwrite Protection (SEHOP)
- EMET - Mandatory ASLR
- EMET - ROP (detects stack pivot)
- EMET - NullPage (farewell null-ptr deref exploits)
- EMET - Export Address Table Filtering (hw bp on EAT)
- Control Flow Guard - breaks vtable overwrites to ROP

Pre Windows 10
- EMET - DEP
- EMET - Structured Exception Handling Overwrite Protection (SEHOP)
- EMET - Mandatory ASLR
- EMET - ROP (detects stack pivot)
- EMET - NullPage (farewell null-ptr deref exploits)
- EMET - Export Address Table Filtering (hw bp on EAT)
- Control Flow Guard - breaks vtable overwrites to ROP

EMET isn't officially supported on 1703+ and hits EOL July 31, 2018

Windows 10 1703

- EMET built in
- Better Control Flow Guard (Export Suppression)
- No child processes
- Disallow win32k syscalls
- Signing policies (DeviceGuard, UMCI)
- Code Integrity Guard
- Arbitrary Code Guard
- Return-Flow Guard??

Windows 10 1703
- EMET built in
- Better Control Flow Guard (Export Suppression)
- No child processes
- Disallow win32k syscalls
- Signing policies (DeviceGuard, UMCI)
- Code Integrity Guard
- Arbitrary Code Guard
- Return-Flow Guard??

Why do we care about CIG/ACG?

- Modular exploitation payloads
- Code injection

Why do we care about CIG/ACG?
Modular exploitation payloads
Code injection

How to enable CIG

There are three ways to enable CIG, in order of increasing precedence.

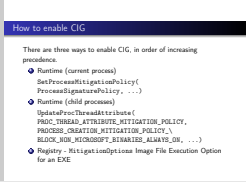
1. Runtime (current process)

   `SetProcessMitigationPolicy( ProcessSignaturePolicy, ...)`

2. Runtime (child processes)

   ```
   UpdateProcThreadAttribute(
   PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY,
   PROCESS_CREATION_MITIGATION_POLICY_\
   BLOCK_NON_MICROSOFT_BINARIES_ALWAYS_ON, ...)
   ```

3. Registry - `MitigationOptions` Image File Execution Option for an EXE



- Once you enable CIG, you can't disable it. Hence the priorities.
- UpdateProcThreadAttribute uses a 4-bit mask to allow defer (0), ALWAYS_ON (1), ALWAYS_OFF (2), and ALWAYS_ON_ALLOW_OPT_OUT (3).
- `BLOCK_NON_MICOSOFT_BINARIES = 1 « 44`
- IFEO at `HKLM/SOFTWARE/Microsoft/Windows NT/CurrentVersion/Image File Execution Options/EXE` - Often used to set the "Debugger" value to windbg, vs, etc.
- This policy is incompatible with third-party edge extensions.

How to enable ACG

There are three ways to enable ACG, in order of increasing precedence.

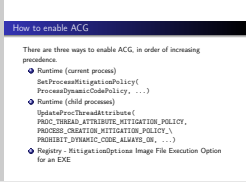
1. Runtime (current process)

   ```
   SetProcessMitigationPolicy(
   ProcessDynamicCodePolicy, ...)
   ```

2. Runtime (child processes)

   ```
   UpdateProcThreadAttribute(
   PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY,
   PROCESS_CREATION_MITIGATION_POLICY_\
   PROHIBIT_DYNAMIC_CODE_ALWAYS_ON, ...)
   ```

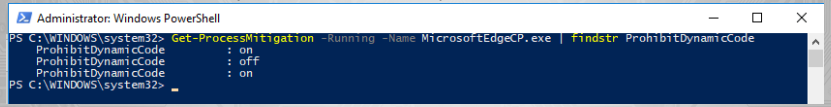
3. Registry - `MitigationOptions` Image File Execution Option for an EXE



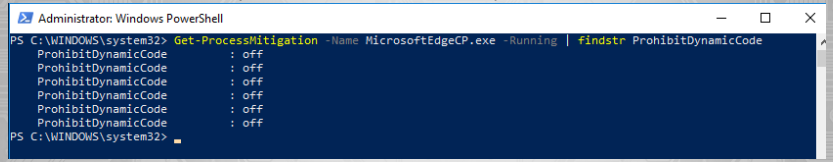
- Can't disable ACG after enabling
- `PROHIBIT_DYNAMIC_CODE = 1 « 36`
- Some graphics drivers may need to turn of ACG. Didn't look into this too deeply.
- SetProcessMitigationPolicy also allows you to specify thread opt-out policy.

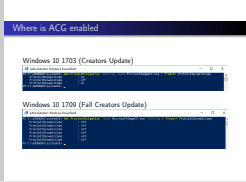
Where is ACG enabled

Windows 10 1703 (Creators Update)



Windows 10 1709 (Fall Creators Update)

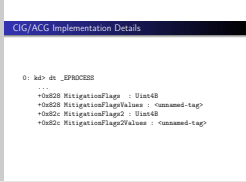




- My kernel symbols were messed up for 1703 - no type info - so all testing and code was run on 1709.

CIG/ACG Implementation Details

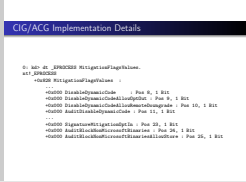
```
0: kd> dt _EPROCESS
    ...
    +0x828 MitigationFlags  : Uint4B
    +0x828 MitigationFlagsValues : <unnamed-tag>
    +0x82c MitigationFlags2 : Uint4B
    +0x82c MitigationFlags2Values : <unnamed-tag>
```

- Since CIG/ACG has to track state somewhere, it's probably located in the EPROCESS.
- We find the MitigationFlags at the end of the EPROCESS structure.
- If we set a write breakpoint on these bitmasks and call SetProcessMitigationPolicy, we can get a stack trace.
- AuditDisableDynamicCode goes to ETW tracing

CIG/ACG Implementation Details

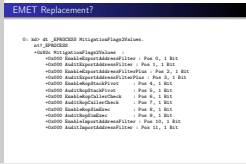
```
0: kd> dt _EPROCESS MitigationFlagsValues.
nt!_EPROCESS
    +0x828 MitigationFlagsValues  :
        ...
        +0x000 DisableDynamicCode     : Pos 8, 1 Bit
        +0x000 DisableDynamicCodeAllowOptOut : Pos 9, 1 Bit
        +0x000 DisableDynamicCodeAllowRemoteDowngrade : Pos 10, 1 Bit
        +0x000 AuditDisableDynamicCode : Pos 11, 1 Bit
        ...
        +0x000 SignatureMitigationOptIn : Pos 23, 1 Bit
        +0x000 AuditBlockNonMicrosoftBinaries : Pos 24, 1 Bit
        +0x000 AuditBlockNonMicrosoftBinariesAllowStore : Pos 25, 1 Bit
```

- AuditDisableDynamicCode goes to ETW tracing
- There are also flags for CFG, high-entropy ASLR, Win32k syscall disallow and filtering, LowIL image mapping, remote image mapping - all features introduced for edge security.

EMET Replacement?

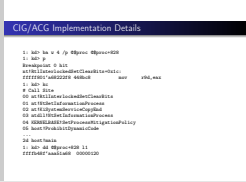
```
0: kd> dt _EPROCESS MitigationFlags2Values.
     nt!_EPROCESS
     +0x82c MitigationFlags2Values  :
         +0x000 EnableExportAddressFilter : Pos 0, 1 Bit
         +0x000 AuditExportAddressFilter : Pos 1, 1 Bit
         +0x000 EnableExportAddressFilterPlus : Pos 2, 1 Bit
         +0x000 AuditExportAddressFilterPlus : Pos 3, 1 Bit
         +0x000 EnableRopStackPivot     : Pos 4, 1 Bit
         +0x000 AuditRopStackPivot      : Pos 5, 1 Bit
         +0x000 EnableRopCallerCheck    : Pos 6, 1 Bit
         +0x000 AuditRopCallerCheck     : Pos 7, 1 Bit
         +0x000 EnableRopSimExec        : Pos 8, 1 Bit
         +0x000 AuditRopSimExec         : Pos 9, 1 Bit
         +0x000 EnableImportAddressFilter : Pos 10, 1 Bit
         +0x000 AuditImportAddressFilter : Pos 11, 1 Bit
```

- Interesting fact - these features all used to be a part of EMET.

CIG/ACG Implementation Details

```
1: kd> ba w 4 /p @$proc @$proc+828
1: kd> p
Breakpoint 0 hit
nt!RtlInterlockedSetClearBits+0x1c:
fffff801`a68222f8 448bc8          mov     r9d,eax
1: kd> kc
# Call Site
00 nt!RtlInterlockedSetClearBits
01 nt!NtSetInformationProcess
02 nt!KiSystemServiceCopyEnd
03 ntdll!NtSetInformationProcess
04 KERNELBASE!SetProcessMitigationPolicy
05 host!ProhibitDynamicCode
...
2d host!main
1: kd> dd @$proc+828 l1
ffffb48f`aaa51a68  00000120
```

- We could reverse SetProcessMitigationPolicy, or just do it live and hope we guessed right.
- Here we see a stack trace of calling SetProcessMitigationPolicy(ACG)
- Boils down to SetInformationProcess to set the 8th bit in MitigationsFlags
- Uses NtSetInformationProcess with undocumented ProcessInformationClass of 0x34

ACG from Usermode

ACG (kernel enforced $W \oplus X$) + CIG

- Code is immutable
- Data cannot become code

Related memory manager APIs should now fail with `ERROR_DYNAMIC_CODE_BLOCKED`

```
VirtualProctect(CodePage, ..., PAGE_READWRITE)
VirtualProtect(DataPage, ..., PAGE_EXECUTE*)
VirtualAlloc(CodePage, ..., PAGE_EXECUTE*)
MapViewOfFile(hMapping, FILE_MAP_EXECUTE, ...)
...
```


ACG from Usermode
ACG (kernel enforced $W \oplus X$) + CIG
- Code is immutable
- Data cannot become code

Related memory manager APIs should now fail with ERROR_DYNAMIC_CODE_BLOCKED

```
VirtualProctect(CodePage, ..., PAGE_READWRITE)
VirtualProtect(DataPage, ..., PAGE_EXECUTE*)
VirtualAlloc(CodePage, ..., PAGE_EXECUTE*)
MapViewOfFile(hMapping, FILE_MAP_EXECUTE, ...)
...
```

- All JIT heaps are now out of process (OOP)

ACG Disable

Why don't we just turn it off?

```
SetProcessMitigationPolicy(
    ProcessDynamicCodePolicy, ...)
```


ACG Disable
Why don't we just turn it off?
SetProcessMitigationPolicy(
ProcessDynamicCodePolicy, ...)

- Didn't dig too far into NtSetInformationProcess, but I assume there's a check to ensure we aren't lessening privileges.
- If there's a way to generate traces in the kernel for certain threads, that would make this much easier - good project idea.
- UpdateProcThreadAttributes states: The specified policy overrides the policies set for the application and the system and cannot be changed after the child process starts running. for PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY
- SetProcessMitigationPolicy states: ASLR mitigation policies cannot be made less restrictive after they have been applied. Seems to apply to ACG as well...

ACG Disable

Why don't we just turn it off?

```
SetProcessMitigationPolicy(
    ProcessDynamicCodePolicy, ...)
```

ERROR_ACCESS_DENIED

ACG Disable
Why don't we just turn it off?
SetProcessMitigationPolicy(
ProcessDynamicCodePolicy, ...)
ERROR_ACCESS_DENIED

- Didn't dig too far into NtSetInformationProcess, but I assume there's a check to ensure we aren't lessening privileges.
- If there's a way to generate traces in the kernel for certain threads, that would make this much easier - good project idea.
- UpdateProcThreadAttributes states: The specified policy overrides the policies set for the application and the system and cannot be changed after the child process starts running. for PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY
- SetProcessMitigationPolicy states: ASLR mitigation policies cannot be made less restrictive after they have been applied. Seems to apply to ACG as well...

ACG VirtualProtect

VirtualProtect a code page to RWX?

```
VirtualProtect(ThisFunction, PAGE_EXECUTE_READWRITE)
```


ACG VirtualProtect
VirtualProtect a code page to RWX?
VirtualProtect(ThisFunction, PAGE_EXECUTE_READWRITE)

- Why does this fail?
- We can use a read breakpoint on MitigationFlagsValues
- NtProtectVirutalMemory - MmProtectVirtualMemory - MiProtectVirtualMemory - MiAllowProtectionChange - MiArbitraryCodeBlocked
- MiArbitraryCodeBlocked checks CurrentThread.DisableDynamicCodeOptOut and Process.DisableDynamicCode
- MiAllowProtectionChange only invoked if MM_EXECUTE_READ is in the access mask

ACG VirtualProtect

VirtualProtect a code page to RWX?

```
VirtualProtect(ThisFunction, PAGE_EXECUTE_READWRITE)
```

ERROR_DYNAMIC_CODE_BLOCKED

ACG VirtualProtect
VirtualProtect a code page to RWX?
VirtualProtect(ThisFunction, PAGE_EXECUTE_READWRITE)
ERROR_DYNAMIC_CODE_BLOCKED

- Why does this fail?
- We can use a read breakpoint on MitigationFlagsValues
- NtProtectVirutalMemory - MmProtectVirtualMemory - MiProtectVirtualMemory - MiAllowProtectionChange - MiArbitraryCodeBlocked
- MiArbitraryCodeBlocked checks CurrentThread.DisableDynamicCodeOptOut and Process.DisableDynamicCode
- MiAllowProtectionChange only invoked if MM_EXECUTE_READ is in the access mask

ACG VirtualProtect

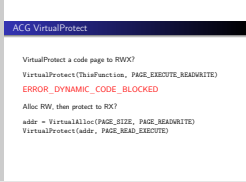
VirtualProtect a code page to RWX?

```
VirtualProtect(ThisFunction, PAGE_EXECUTE_READWRITE)
```

ERROR_DYNAMIC_CODE_BLOCKED

Alloc RW, then protect to RX?

```
addr = VirtualAlloc(PAGE_SIZE, PAGE_READWRITE)
VirtualProtect(addr, PAGE_READ_EXECUTE)
```

- Why does this fail?
- We can use a read breakpoint on MitigationFlagsValues
- NtProtectVirutalMemory - MmProtectVirtualMemory - MiProtectVirtualMemory - MiAllowProtectionChange - MiArbitraryCodeBlocked
- MiArbitraryCodeBlocked checks CurrentThread.DisableDynamicCodeOptOut and Process.DisableDynamicCode
- MiAllowProtectionChange only invoked if MM_EXECUTE_READ is in the access mask

ACG VirtualProtect

VirtualProtect a code page to RWX?

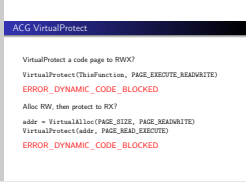
```
VirtualProtect(ThisFunction, PAGE_EXECUTE_READWRITE)
```

ERROR_DYNAMIC_CODE_BLOCKED

Alloc RW, then protect to RX?

```
addr = VirtualAlloc(PAGE_SIZE, PAGE_READWRITE)
VirtualProtect(addr, PAGE_READ_EXECUTE)
```

ERROR_DYNAMIC_CODE_BLOCKED



- Why does this fail?
- We can use a read breakpoint on MitigationFlagsValues
- NtProtectVirutalMemory - MmProtectVirtualMemory - MiProtectVirtualMemory - MiAllowProtectionChange - MiArbitraryCodeBlocked
- MiArbitraryCodeBlocked checks CurrentThread.DisableDynamicCodeOptOut and Process.DisableDynamicCode
- MiAllowProtectionChange only invoked if MM_EXECUTE_READ is in the access mask

ACG VirtualAlloc

```
VirtualAlloc(PAGE_SIZE, PAGE_EXECUTE_READWRITE)
```

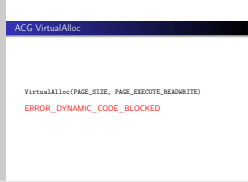

ACG VirtualAlloc
VirtualAlloc(PAGE_SIZE, PAGE_EXECUTE_READWRITE)

- Why does this fail?
- NtAllocateVirtualMemory - MiAllocateVirtualMemory - MiArbitraryCodeBlocked
- Same issue as VirtualProtect

ACG VirtualAlloc

`VirtualAlloc(PAGE_SIZE, PAGE_EXECUTE_READWRITE)`

ERROR_DYNAMIC_CODE_BLOCKED

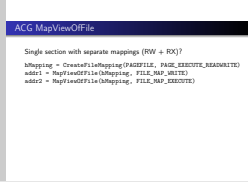


- Why does this fail?
- NtAllocateVirtualMemory - MiAllocateVirtualMemory - MiArbitraryCodeBlocked
- Same issue as VirtualProtect

ACG MapViewOfFile

Single section with separate mappings (RW + RX)?

```
hMapping = CreateFileMapping(PAGEFILE, PAGE_EXECUTE_READWRITE)
addr1 = MapViewOfFile(hMapping, FILE_MAP_WRITE)
addr2 = MapViewOfFile(hMapping, FILE_MAP_EXECUTE)
```

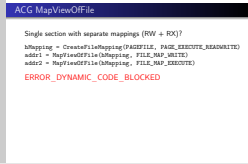
- If we create the file mapping with PAGE_READWRITE, trying to map it as execute will result in ERROR_INVALID_PARAMETER.
- MiArbitraryCodeBlocked referenced in MiMapViewOfSection and MiMapViewOfImageSection
- We really can't do much from usermode except for invoking LoadLibrary - which is protected by CIG. We're left with looking at other interfaces for post-exploitation.

ACG MapViewOfFile

Single section with separate mappings (RW + RX)?

```
hMapping = CreateFileMapping(PAGEFILE, PAGE_EXECUTE_READWRITE)
addr1 = MapViewOfFile(hMapping, FILE_MAP_WRITE)
addr2 = MapViewOfFile(hMapping, FILE_MAP_EXECUTE)
```

ERROR_DYNAMIC_CODE_BLOCKED



- If we create the file mapping with PAGE_READWRITE, trying to map it as execute will result in ERROR_INVALID_PARAMETER.
- MiArbitraryCodeBlocked referenced in MiMapViewOfSection and MiMapViewOfImageSection
- We really can't do much from usermode except for invoking LoadLibrary - which is protected by CIG. We're left with looking at other interfaces for post-exploitation.

ACG MapViewOfFile

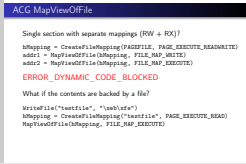
Single section with separate mappings (RW + RX)?

```
hMapping = CreateFileMapping(PAGEFILE, PAGE_EXECUTE_READWRITE)
addr1 = MapViewOfFile(hMapping, FILE_MAP_WRITE)
addr2 = MapViewOfFile(hMapping, FILE_MAP_EXECUTE)
```

ERROR_DYNAMIC_CODE_BLOCKED

What if the contents are backed by a file?

```
WriteFile("testfile", "\xeb\xfe")
hMapping = CreateFileMapping("testfile", PAGE_EXECUTE_READ)
MapViewOfFile(hMapping, FILE_MAP_EXECUTE)
```

- If we create the file mapping with PAGE_READWRITE, trying to map it as execute will result in ERROR_INVALID_PARAMETER.
- MiArbitraryCodeBlocked referenced in MiMapViewOfSection and MiMapViewOfImageSection
- We really can't do much from usermode except for invoking LoadLibrary - which is protected by CIG. We're left with looking at other interfaces for post-exploitation.

ACG MapViewOfFile

Single section with separate mappings (RW + RX)?

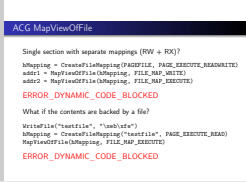
```
hMapping = CreateFileMapping(PAGEFILE, PAGE_EXECUTE_READWRITE)
addr1 = MapViewOfFile(hMapping, FILE_MAP_WRITE)
addr2 = MapViewOfFile(hMapping, FILE_MAP_EXECUTE)
```

ERROR_DYNAMIC_CODE_BLOCKED

What if the contents are backed by a file?

```
WriteFile("testfile", "\xeb\xfe")
hMapping = CreateFileMapping("testfile", PAGE_EXECUTE_READ)
MapViewOfFile(hMapping, FILE_MAP_EXECUTE)
```

ERROR_DYNAMIC_CODE_BLOCKED



- If we create the file mapping with PAGE_READWRITE, trying to map it as execute will result in ERROR_INVALID_PARAMETER.
- MiArbitraryCodeBlocked referenced in MiMapViewOfSection and MiMapViewOfImageSection
- We really can't do much from usermode except for invoking LoadLibrary - which is protected by CIG. We're left with looking at other interfaces for post-exploitation.

ACG from Kernelmode

- Attacking ACG should be much easier from kernelmode
- Need to disable ACG or allocate executable memory on behalf of a userspace loader
- How can we do this?

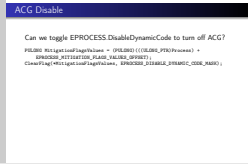
ACG from Kernelmode
- Attacking ACG should be much easier from kernelmode
- Need to disable ACG or allocate executable memory on behalf of a userspace loader
- How can we do this?

- Obviously, Microsoft doesn't consider kernel-mode manipulations bypasses. We still need to consider them for code injection implications.

ACG Disable

Can we toggle EPROCESS.DisableDynamicCode to turn off ACG?

```
PULONG MitigationFlagsValues = (PULONG)(((ULONG_PTR)Process) +
    EPROCESS_MITIGATION_FLAGS_VALUES_OFFSET);
ClearFlag(*MitigationFlagsValues, EPROCESS_DISABLE_DYNAMIC_CODE_MASK);
```

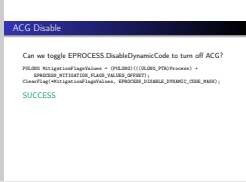
- Drawback - Static per-build constants aren't very maintainable
- Interesting fact - We can opt out individual threads even if EPROCESS.MitigationFlags.DisableDynamicCodeAllowOptOut is FALSE! I guess the EPROCESS OptOut is only checked in SetThreadInformation with ThreadDynamicCodePolicy (Usermode way of setting opt out)

ACG Disable

Can we toggle EPROCESS.DisableDynamicCode to turn off ACG?

```
PULONG MitigationFlagsValues = (PULONG)(((ULONG_PTR)Process) +
    EPROCESS_MITIGATION_FLAGS_VALUES_OFFSET);
ClearFlag(*MitigationFlagsValues, EPROCESS_DISABLE_DYNAMIC_CODE_MASK);
```

SUCCESS



- Drawback - Static per-build constants aren't very maintainable
- Interesting fact - We can opt out individual threads even if EPROCESS.MitigationFlags.DisableDynamicCodeAllowOptOut is FALSE! I guess the EPROCESS OptOut is only checked in SetThreadInformation with ThreadDynamicCodePolicy (Usermode way of setting opt out)

ACG Disable

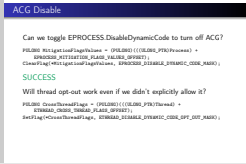
Can we toggle EPROCESS.DisableDynamicCode to turn off ACG?

```
PULONG MitigationFlagsValues = (PULONG)(((ULONG_PTR)Process) +
    EPROCESS_MITIGATION_FLAGS_VALUES_OFFSET);
ClearFlag(*MitigationFlagsValues, EPROCESS_DISABLE_DYNAMIC_CODE_MASK);
```

SUCCESS

Will thread opt-out work even if we didn't explicitly allow it?

```
PULONG CrossThreadFlags = (PULONG)(((ULONG_PTR)Thread) +
    ETHREAD_CROSS_THREAD_FLAGS_OFFSET);
SetFlag(*CrossThreadFlags, ETHREAD_DISABLE_DYNAMIC_CODE_OPT_OUT_MASK);
```

- Drawback - Static per-build constants aren't very maintainable
- Interesting fact - We can opt out individual threads even if EPROCESS.MitigationFlags.DisableDynamicCodeAllowOptOut is FALSE! I guess the EPROCESS OptOut is only checked in SetThreadInformation with ThreadDynamicCodePolicy (Usermode way of setting opt out)

ACG Disable

Can we toggle EPROCESS.DisableDynamicCode to turn off ACG?

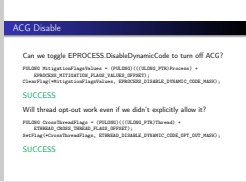
```
PULONG MitigationFlagsValues = (PULONG)(((ULONG_PTR)Process) +
    EPROCESS_MITIGATION_FLAGS_VALUES_OFFSET);
ClearFlag(*MitigationFlagsValues, EPROCESS_DISABLE_DYNAMIC_CODE_MASK);
```

SUCCESS

Will thread opt-out work even if we didn't explicitly allow it?

```
PULONG CrossThreadFlags = (PULONG)(((ULONG_PTR)Thread) +
    ETHREAD_CROSS_THREAD_FLAGS_OFFSET);
SetFlag(*CrossThreadFlags, ETHREAD_DISABLE_DYNAMIC_CODE_OPT_OUT_MASK);
```

SUCCESS



- Drawback - Static per-build constants aren't very maintainable
- Interesting fact - We can opt out individual threads even if EPROCESS.MitigationFlags.DisableDynamicCodeAllowOptOut is FALSE! I guess the EPROCESS OptOut is only checked in SetThreadInformation with ThreadDynamicCodePolicy (Usermode way of setting opt out)

ACG NtProtectVirtualMemory or ZwAllocateVirtualMemory

- No ZwProtectVirtualMemory, NtProtectVirtualMemory and MmProtectVirtualMemory aren't exported.
- `ZwAllocateVirtualMemory(PAGE_EXECUTE_READWRITE)` fails with STATUS_DYNAMIC_CODE_BLOCKED

ACG NtProtectVirtualMemory or ZwAllocateVirtualMemory
- No ZwProtectVirtualMemory, NtProtectVirtualMemory and MmProtectVirtualMemory aren't exported.
- ZwAllocateVirtualMemory(PAGE_EXECUTE_READWRITE) fails with STATUS_DYNAMIC_CODE_BLOCKED

- NtProtectVirtualMemory or MmProtectVirtualMemory would probably fail as well even if you could find their addresses.

ACG Memory Descriptor List (MDL)

Use an MDL to map RX memory into System space and write shellcode

```
mdl = MmProbeAndLockPages(UserAddress, Size)
SystemAddr = MmGetSystemAddressForMdlSafe(mdl)
memcpy(SystemAddr, shellcode)
```


ACG Memory Descriptor List (MDL)
Use an MDL to map RX memory into System space and write shellcode
mdl = MmProbeAndLockPages(UserAddress, Size)
SystemAddr = MmGetSystemAddressForMdlSafe(mdl)
memcpy(SystemAddr, shellcode)

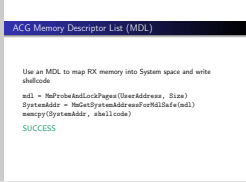
- Unfortunately we're limited to modifying existing code pages. This may be enough, but it doesn't really give us the arbitrary code primitive we're looking for.

ACG Memory Descriptor List (MDL)

Use an MDL to map RX memory into System space and write shellcode

```
mdl = MmProbeAndLockPages(UserAddress, Size)
SystemAddr = MmGetSystemAddressForMdlSafe(mdl)
memcpy(SystemAddr, shellcode)
```

SUCCESS



- Unfortunately we're limited to modifying existing code pages. This may be enough, but it doesn't really give us the arbitrary code primitive we're looking for.

ACG Toggle Write Protect

- System-wide write protection can be enabled/disabled through cr0 bit 16.
- Toggle this to allow us to write shellcode to userspace RX memory?

```
cr0 = __readcr0();
ClearFlag(cr0, CR0_WRITE_PROTECT_MASK);
__writecr0(cr0);
memcpy(Address, Buffer, Size);
```


ACG Toggle Write Protect
- System-wide write protection can be enabled/disabled through cr0 bit 16.
- Toggle this to allow us to write shellcode to userspace RX memory?

```
cr0 = __readcr0();
ClearFlag(cr0, CR0_WRITE_PROTECT_MASK);
__writecr0(cr0);
memcpy(Address, Buffer, Size);
```

- The processor won't fault when we try to write to a VA that isn't writable! Ignores r/w bit in PTE in ring 0
- Can only modify existing code pages
- Unfortunately PatchGuard will catch this. and likely throw a CRITICAL_STRUCTURE_CORRUPTION bugcheck with corrupted type of 15 - A processor control register.

ACG Toggle Write Protect

- System-wide write protection can be enabled/disabled through cr0 bit 16.
- Toggle this to allow us to write shellcode to userspace RX memory?

```
cr0 = __readcr0();
ClearFlag(cr0, CR0_WRITE_PROTECT_MASK);
__writecr0(cr0);
memcpy(Address, Buffer, Size);
```

SUCCESS

ACG Toggle Write Protect
System-wide write protection can be enabled/disabled through cr0 bit 16.
Toggle this to allow us to write shellcode to userspace RX memory?
cr0 = __readcr0();
ClearFlag(cr0, CR0_WRITE_PROTECT_MASK);
__writecr0(cr0);
memcpy(Address, Buffer, Size);
SUCCESS

- The processor won't fault when we try to write to a VA that isn't writable! Ignores r/w bit in PTE in ring 0
- Can only modify existing code pages
- Unfortunately PatchGuard will catch this. and likely throw a CRITICAL_STRUCTURE_CORRUPTION bugcheck with corrupted type of 15 - A processor control register.

ACG PTE Manipulation

- What if we start manipulating paging structures directly?
- Read page table base from cr3, and traverse physical pages (beats KASLR) to get to a PTE.
- Disable NoExecute bit to get a RWX page.

ACG PTE Manipulation
- What if we start manipulating paging structures directly?
- Read page table base from cr3, and traverse physical pages (beats KASLR) to get to a PTE.
- Disable NoExecute bit to get a RWX page.

- No code snippet for this one. Lots of bit shifting and masking. Code will be available for download.
- Bypasses pretty much all mitigations, no major drawbacks
- Fairly universal, since hardware doesn't change too often.

ACG PTE Manipulation

- What if we start manipulating paging structures directly?
- Read page table base from cr3, and traverse physical pages (beats KASLR) to get to a PTE.
- Disable NoExecute bit to get a RWX page.

SUCCESS

ACG PTE Manipulation
- What if we start manipulating paging structures directly?
- Read page table base from cr3, and traverse physical pages (beats KASLR) to get to a PTE.
- Disable NoExecute bit to get a RWX page.

SUCCESS

- No code snippet for this one. Lots of bit shifting and masking. Code will be available for download.
- Bypasses pretty much all mitigations, no major drawbacks
- Fairly universal, since hardware doesn't change too often.

Summary

Microsoft has done a lot to make initial exploitation harder. Even if you have bugs they may not all be exploitable in a modular way.

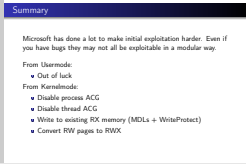
From Usermode:

- Out of luck

From Kernelmode:

- Disable process ACG
- Disable thread ACG
- Write to existing RX memory (MDLs + WriteProtect)
- Convert RW pages to RWX

└Summary



References

David Weston and Matt Miller.
Microsoft's strategy and technology improvements toward mitigating arbitrary native code execution.

Swamy Shivaganga Nagaraju.
Agility with security mitigations in windows 10.

References
David Weston and Matt Miller.
Microsoft's strategy and technology improvements toward mitigating arbitrary native code execution.
Swamy Shivaganga Nagaraju.
Agility with security mitigations in windows 10.

- Both presentations were given in MAR 2017 shortly after original Creators Update was released. One at NullCon and the other at CanSecWest.